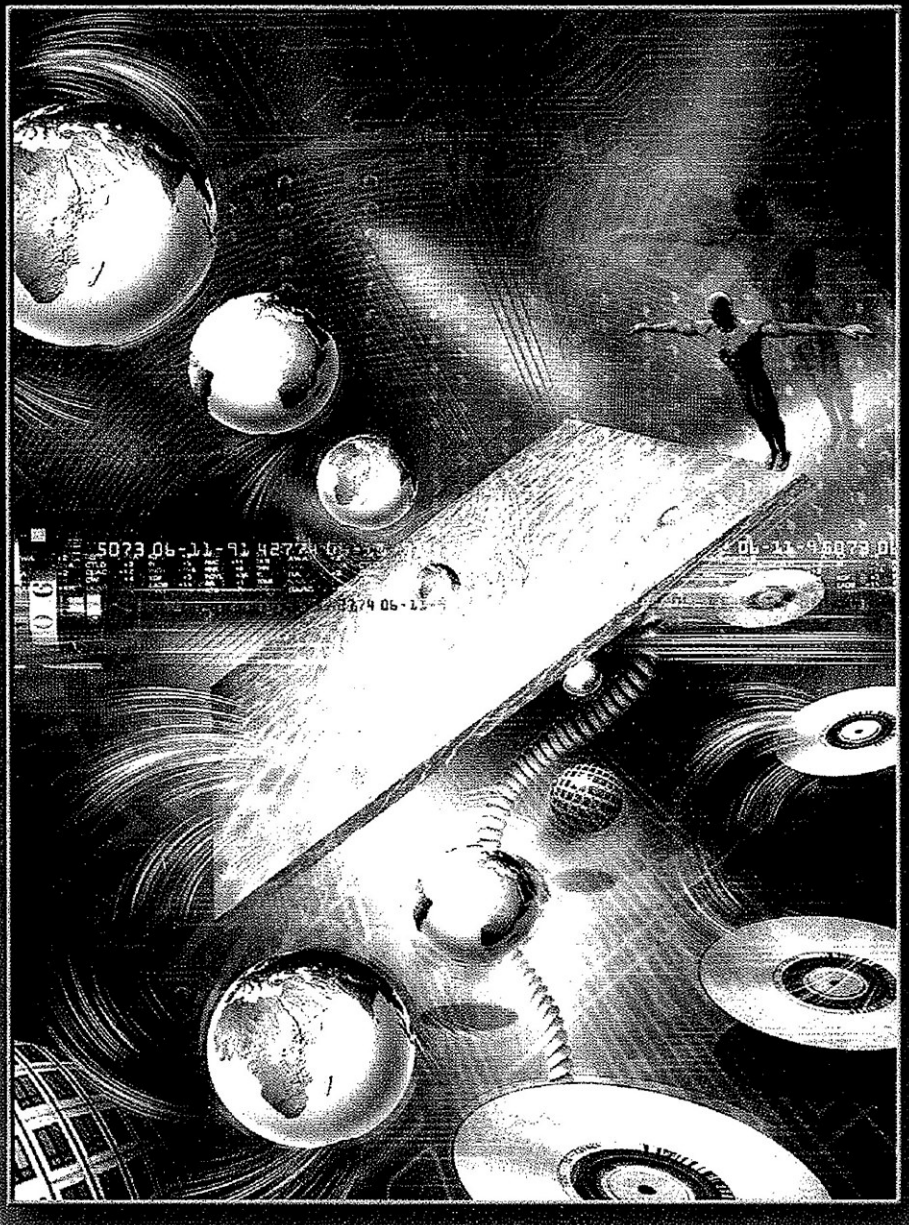
COMPUTERBOOKS

Hatvany Béla Csaba

ASP 3.0 programozás



Hatvany Béla Csaba

ASP 3.0 programozás

LEKTOR
Fejes Balázs

COMPUTERBOOKS
BUDAPEST, 2004

Tartalomjegyzék

Előszó 1

1. Ismerkedés az ASP technológiával 3
1.1 A HTML-től az ASP-ig - a webszerverek három generációja 3
1.1.1 CGI: a webszerverek első generációja 3
1.1.2 ISAPI/NSAPI: a webszerverek második generációja 4
1.1.3 ASP: a webszerverek harmadik generációja 5
1.2 Az IIS és a webalkalmazások 7
1.2.1 Az Internet Information Services telepítése és beállítása 7
1.2.3 Webalkalmazások és munkamenetek 14
1.2.4 Virtuális mappák és alkalmazások létrehozása 17
1.2.5 A könyvben szereplő példákról 23
1.3 Miként működik? 29
1.3.1 Egy ASP lap felépítése és végrehajtása 29
1.3.2 Szerver- és kliensoldali szkriptek 33

2. Az ASP belső objektumai 41
2.1 A `global.asa` állomány 41
2.2 A `Server` objektum 43
2.2.1 A `Server` objektum metódusai 43
2.2.2 A `ScriptTimeout` tulajdonság 53
2.3 Az `Application` objektum 54
2.3.1 Az `Application` objektum eseményei 54
2.3.2 Az `Application` objektum gyűjteményei 60
2.3.3 Az `Application` objektum metódusai 63
2.4 A `Session` objektum 64
2.4.1 A `Session` objektum eseményei 65
2.4.2 A `Session` objektum tulajdonságai 68
2.4.3 A `Session` objektum gyűjteményei 71
2.4.4 A `Session` objektum `Abandon` metódusa 73
2.5 A `Request` objektum 76
2.5.1 A `Request` objektum `BinaryRead` metódusa és `TotalBytes` tulajdonsága 76
2.5.2 A `Request` objektum gyűjteményei 80
2.6 A `Response` objektum 91
2.6.1 A `Response` objektum metódusai 92
2.6.2 A `Response` objektum tulajdonságai 102
2.7 A `Response` és `Request` objektum `Cookies` gyűjteménye 105

2.8 Az ASPError objektum 113
2.8.1 Az ASPError objektum tulajdonságai 114
2.8.2 Egyéni hibajelentő lapok elkészítése 116

3. A Scripting objektumok 119
3.1 A FileSystemObject (FSO) objektum 120
3.1.1 A FileSystemObject objektum metódusai 122
3.1.2 A FileSystemObject objektum Drives tulajdonsága 133
3.2 A Drive objektum 133
3.2.1 A Drive objektum tulajdonságai 134
3.3 A Folder objektum 138
3.3.1 A Folder objektum metódusai 138
3.3.2 A Folder objektum tulajdonságai 140
3.4 A File objektum 146
3.4.1 A File objektum metódusai 146
3.4.2 A File objektum tulajdonságai 150
3.5 A TextStream objektum 155
3.5.1 A TextStream objektum metódusai és tulajdonságai 156
3.6 A Dictionary objektum 159
3.6.1 A Dictionary objektum metódusai és tulajdonságai 160

4. MS web- és email-komponensek 165
4.1 Az AdRotator (hirdetésforgató) komponens 167
4.2 A ContentRotator (tartalomforgató) komponens 173
4.3 A BrowserType (a böngésző képességeit megadó) komponens 175
4.4 A NextLink (tartalomkapcsoló) komponens 182
4.5 A Counters (számlálókat kezelő) komponens 187
4.6 A IISLOG (naplókezelő) komponens 190
4.7 A MyInfo (egyedi információt kezelő) komponens 196
4.8 A PageCounter (lapszámláló) komponens 200
4.9 A PermissionChecker (jogosultság-ellenőrző) komponens 202
4.10 A Tools (eszköztár) komponens 205
4.11 A CDONTS email-komponens 208
4.12 A CDOSYS email-komponens 211

5. ASP komponensek előállítása 215
5.1 A COM (*Component Object Model*) technológia rövid áttekintése 215
5.2 Szkript komponensek 218
5.2.1 A feladat és megoldása WSH szkripttel 219
5.2.2 A feladat megoldása VBScript osztállyal 222
5.2.3 A Scriptlet Wizard 226
5.2.4 A szkript komponens felépítése 230
5.2.5 A szkript komponens működése 233

5.2.6 A szkript komponens ellenőrzése Visual Basic programmal 238
5.2.7 A szkript komponens ASP alkalmazása 241
5.3 Visual Basic komponensek 242
5.3.1 A Visual Basic komponensek előállítása 243
5.3.2 A Visual Basic komponens alkalmazása 249
5.4 Visual Basic ASP komponensek 255

6. Jelszóval védett weblapok - egy példaalkalmazás előállítása 265
6.1 A feladat meghatározása és előkészületek 265
6.2 A kezdő és a regisztráló lap 267
6.3 A beadott adatok ellenőrzése 270
6.4 Új tag regisztrálása 273
6.5 Az email megszerkesztése és elküldése 279
6.6 Az adatok módosítása és törlése, a jelszó lekérdezése 282
6.7 Az állományok megvédése 287
6.8 A tagok listája 288
6.9 A hírlevelek elküldése 297
6.10 Befejező megjegyzések 303

Hogyan tovább? 305

Irodalomjegyzék 307

Tárgymutató 309

Előszó

Kilenc évvel ezelőtt, 1994 októberében Tim Breners-Lee, a web feltalálója, létrehozta a W3C konzorciumot (World Wide Web Consortium, www.w3.org). Azóta a webhelyek *(web site)* a kezdeti statikus HTML lapok gyűjteményétől egy hosszú és eredményes fejlődés során eljutottak a napjainkban ismert webes alkalmazásokhoz.

Manapság egy webhelytől joggal elvárjuk, hogy dinamikus és adatvezérelt (*data driven*) webalkalmazás alakjában jelenítse meg a felkínált információt a böngészőnkben. Ilyen például az Amazon.com (www.amazon.com) vagy az eBay.com (www.ebay.com) és számtalan más webhely is. Megszoktuk, hogy a webalkalmazásokkal valóságos párbeszédet folytathatunk. Ha egyelőre a webszerveren futó alkalmazások nem rendelkeznek még a hagyományos lokális alkalmazások összes tulajdonságával, a fejlesztésükre rendelkezésre álló technológia egyre több lehetőséget biztosít a fejlesztőnek, hogy a két alkalmazástípus közötti különbséget minimálisra csökkentse.

A webalkalmazások fejlesztésére több technológia áll rendelkezésünkre. Ez a könyv a Microsoft Active Server Pages (ASP) technológiával foglalkozik, pontosabban az ASP 3.0 verziójával. A könyv bevezeti az olvasót az ASP 3.0 webtechnologiába és végigvezeti ennek a technológiának a főbb fejezetein. A könyv elolvasása után a kedves Olvasó képes lesz bonyolultabb ASP alkalmazásokat is fejleszteni.

Az első fejezetben az Olvasó megismerkedik az ASP technológia keletkezésével, feladataival és a vázlatos felépítésével. Ugyanitt arról is értesül, hogy mire van szüksége egy ASP alkalmazás fejlesztésére és megismerkedik a legelső egyszerű ASP alkalmazással.

ASP alkalmazásokat több szkriptnyelven lehet fejleszteni. A legelterjedtebbek ezek közül a VBScript és a JScript, kettőjük közül pedig az első. Ezért, a könyvben a példákat a VBScrip nyelv segítségével mutatom be. Feltételezem, hogy a könyv olvasója ismeri legalább egyikét ezeknek a nyelveknek.

A második fejezettel elkezdem az ASP technológia rendszeres leírását. Ebben a fejezetben az ASP belső objektumait és ezek tulajdonságait és metódusait ismerhetjük meg.

A harmadik fejezet a Scripting objektumokkal foglakozik, melyek az alapvető állományműveleteket teszik lehetővé.

A hagyományos lokális Windows alkalmazások fejlesztésében fontos szerepet játszanak azok a kódrészletek, amelyek általános feladatokat oldanak meg és amelyeket is-

mételten, több programba is felhasználhatunk: az úgynevezett COM komponensek. Ezek szerepét az ASP technológiában az ASP komponensek játsszák.

A negyedik fejezet az MS ASP komponensekkel foglakozik. Többek között, itt megismerkedünk a lapok közötti navigációt, a webhirdetést és az email üzenetek elküldését megkönnyítő komponensekkel.

Sok ASP komponenst írtak és írnak még mindig a legkülönbözőbb webes feladatok megoldására. Nyugodtan állíthatjuk, hogy gyártásukkal egy egész új szoftverszolgáltatás foglalkozik. Mindennek ellenére legalább három okunk van arra, hogy tudjuk, miként állítható elő egy ASP komponens. Legelőször is, könnyen előfordulhat, hogy egy nagyon speciális komponensre van szükségünk, amit a kínálat nem tartalmaz és azt saját magunknak kell kifejlesztenünk. Másodszor, kisebb komponensek esetében olcsóbb, ha saját magunk írjuk meg, mint hogy ha megvennénk. Harmadszor pedig, ha a komponensek előállítása ilyen jó üzlet, akkor tényleg érdemes tudni, hogy kell ASP komponenseket megírni. Az ötödik fejezet az ASP komponensekkel foglalkozik.

Az ASP technológia több rétegből vagy programozási interfészből áll. A könyv korlátozott terjedelme nem teszi lehetővé, hogy a szűk értelemben vett ASP-n kívül, azaz az alaprétegen kívül, más ASP összetevőkkel is foglalkozzunk. Ezért, ebben a könyvben nem esik szó például az ADO-ról (ActiveX Data Objects) vagy az XML-ről (Extensible Markup Language). Mivelhogy aligha lehet egy komolyabb ASP alkalmazást adatbázis nélkül elképzelni, az utolsó fejezet egy olyan alkalmazás részletes leírását tartalmazza, melynek adatállományát egy adatbázis ellenőrzi. Az alkalmazás megértéséhez az Olvasónak ismerni kell az adatbázis-kezelés alapjait.

A könyv több mint 100 példaalkalmazással szemlélteti a bemutatott anyagot. A webhelyemen (http://www.hatvany-online.net/aspclassic) a kedves Olvasó a szemléltető anyagot úgy VBScrip, mint JScript változatban *online* kipróbálhatja, illetve a teljes forráskódot, amelyet az asp30.zip állományba tömörítettem, letöltheti.

A könyv keletkezése folyamán a szemléltető példákat elkészítésük után folyamatosan tettem fel a webhelyemre. Sok látogatómtól értékes megjegyzéseket és észrevételeket kaptam. Nekik, Fejes Balázsnak (MCP, MCSD) a könyv lektorának, Benkő Tibornénak, aki a kéziratot az irodalmi nyelvhez igazította, valamint mindazoknak, akik valamilyen formában hozzájárultak a könyv megjelenéséhez, ezúton fejezem ki hálás köszönetemet.

Dr. Hatvany Béla Csaba
(csaba@hatvany-online.net)

2003. október 16.
Kuchen, Németország

1. Ismerkedés az ASP technológiával

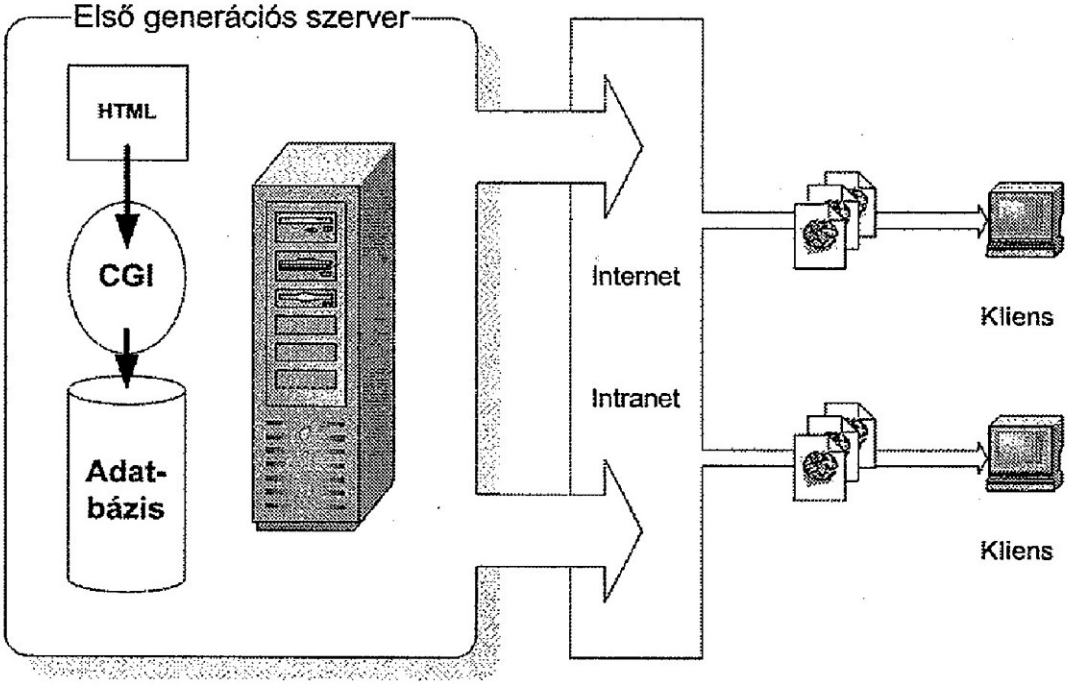
1.1 A HTML-től az ASP-ig - a webszerverek három generációja

Alighogy az első webszerverek működésbe léptek, máris nyilvánvalóvá vált, hogy a statikus weblapok nem felelnek meg a felhasználók igényeinek. Grafikus elemek felhasználásával ugyan nagyon vonzóvá lehetett tenni egy weblapot, a Netscape által bevezetett kliensoldali JavaScript segítségével pedig egy kevés dinamikát is be lehetett vinni a lapokba, a legnagyobb hiányosságot azonban, név szerint az információ kizárólagos egyirányú közlekedését a szervertől a kliens felé, az első szerverek nem voltak képesek áthidalni.

1.1.1 CGI: a webszerverek első generációja

Ezért, még az első szervergenerációban, a fejlesztők nagy hangsúlyt helyeztek az információáramlás javítására: létrejött a **Common Gateway Interface** (CGI). Egy csatlakozó felületről vagy interfészről van szó, mely megengedi, hogy célirányzottan szerveroldali programokat futtassunk. Legtöbbször kisebb C-programokról vagy Perl-szkriptekről van szó, melyek a szerver **cgi-bin** mappájában találhatók. A programokat a webszerver hívja meg és interakciója ezekkel a programokkal az **stdin/stdout** interfészen keresztül jön létre.

Az elsőgenerációs webszerverek felépítését az 1.1. ábrán láthatjuk.



1.1. ábra Első generációs webszerver

A CGI előnye elsősorban abban állt, hogy minden program saját folyamatban futott. A webszerver működését nem befolyásolta az, hogy az egyik vagy a másik CGI-program hibát generált vagy éppenséggel megszakadt. Ezenkívül a CGI-programok könnyen hordozhatók (*porting*) más platformokra, hiszen az webszerverrel történő kommunikáció az **stdin/stdout** interfészen keresztül történik, ez viszont platformfüggetlen.

A CGI interfész nagy hátránya a hatalmas erőforrásétvágya. Mivelhogy minden program saját folyamatában fut, ahányszor egy és ugyanaz a program elindul, ugyanannyiszor indít a rendszer egy új folyamatot. Kevésbé látogatott weblapok esetében a szerver valahogy még megbirkózik a feladattal, gyakran látogatott weblapok viszont teljesen lebéníthatják a webszervert.

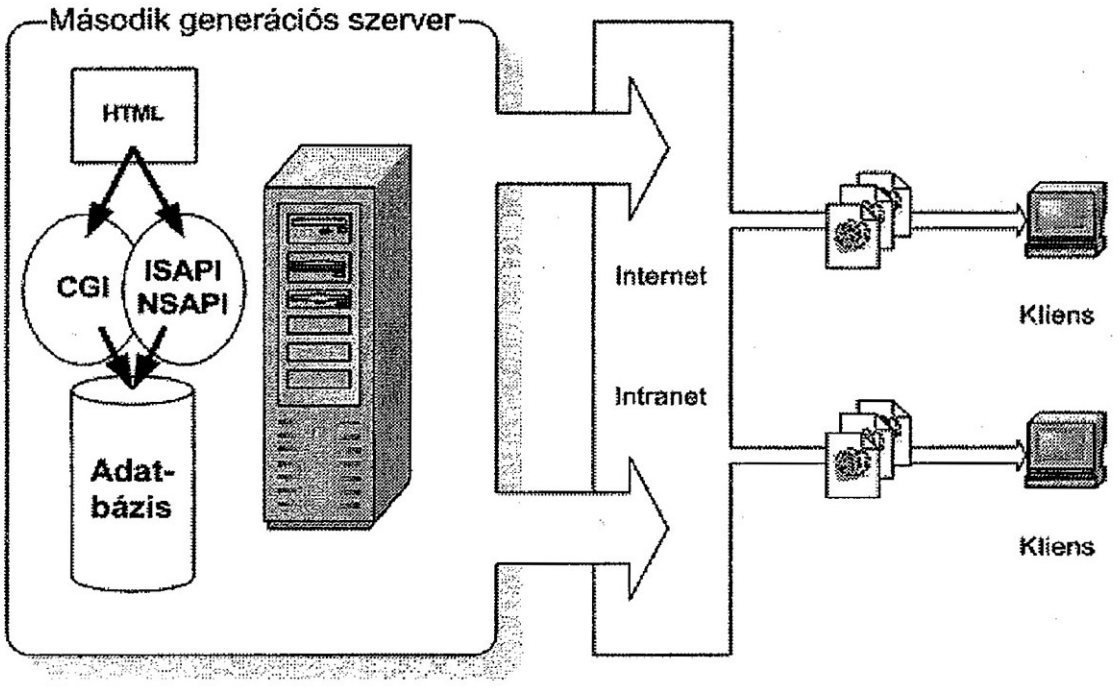
A legelső weblapok egyetemek és kutatóközpontok szerverein kaptak helyet. Nemsokára azonban elkezdték, először félénken, majd minél határozottabban diadalútjukat a vállalatok meghódítására. Egyre több vállalat, ezeken belül leginkább a marketingosztályok, láttak itt új lehetőségeket. Kevés időn belül a weblapok elsöprő többsége vállalati megrendelésre készült. A kliensoldalon végrehajtott JavaScript vagy VBScript adta lehetőségek távolról sem elégítették ki az új felhasználók igényeit. Hiányzott az üzleti világnak oly rendkívül fontos tényező: a biztonságos kétirányú adatközvetítés.

1.1.2 ISAPI/NSAPI: a webszerverek második generációja

Hogy az új megrendelők igényeinek meg tudjanak felelni Microsoft, és vele majdnem egy időben, Netscape elkészítettek saját webszerverük részére egy új interfészt. Microsoft előállt az **Internet Server Application Programming Interface** (ISAPI), Netscape pedig a **Netscape Server Application Programming Interface** (NSAPI) csatlakozó felülettel. Az ISAPI/NSAPI interfészek lehetővé teszik, hogy a kliens által legelőször igényelt program saját folyamatban induljon, majd miután a szerver egy programot elindított, a program addig marad a szerver memóriájában, ameddig a szervert le nem állítják. Ez azt jelenti, hogy egy program csak egyszer fut egy ISAPI/NSAPI szerveren és ugyanaz a program sok lekérdezést tud megválaszolni. Ezzel a webszerverek megterhelése lényegesen csökkent, teljesítményük pedig többszörösre nőtt.

A második generációs webszerverek felépítésének vázát az 1.2. ábrán láthatjuk.

Az ISAPI/NSAPI rutinokat igénybevevő programok bármilyen programozási nyelven írhatók, mely képes a szerveren végrehajtható bináris kódot előállítani. Ilyen programozási nyelv például a C++, a Java vagy a Delphi. Ezzel lényegesen megnőtt a webalkalmazások fejlesztésének lehetősége és ez rövid időn belül szemmel láthatóan megmutatta hatását.



1.2. ábra Második generációs webszerver

1.1.3 ASP: a webszerverek harmadik generációja

Már az első két generációs webszerverek is képesek voltak arra, hogy a kliens kérőüzeneteit kellőképpen feldolgozzák, elemezzék és megválaszolják. A kliens által kitöltött űrlap adatait a kliens böngészője elküldi a webszervernek. A szerver a kapott adatokat továbbküldi egy CGI programnak vagy ISAPI/NSAPI rutinnak, amely kiértékeli az adatokat. A kapott értékektől függően a CGI program vagy az ISAPI/NSAPI rutin akár egy egész HTML lapot is elkészíthet és ez a szervernek küldi. A szerver azonnal továbbítja a lapot a kliensnek.

A HTML alkalmazások bonyolultságával együtt az alkalmazott CGI programok és ISAPI/NSAPI rutinok komplexitása is nőt. Nagyon nehéz volt kliensadatokat hatékonyan és biztonságosan több weblapon keresztül szállítani. Kialakított alkalmazásokon csekély módosítások is hatalmas munkaerő bevetését követelték. A gondot az okozta, hogy a HTML lapok és a velük együttműködő CGI programok vagy ISAPI/NSAPI rutinok között nem volt egy szerves kapcsolat, két különálló világot képviseltek nagyon komplikált kommunikációs csatornával. Új megoldást kellett keresni. Microsoft válasza erre a kihívásra az ASP volt.

Az ASP (**Active Server Pages**) új lehetőségeket kínál a webszervereknek. Az ASP nem egy programozási nyelv, de felhasznál szkriptnyelveket: a VBScript és a JScript szkriptnyelveket. Az ASP nem egy program, de felhasznál előre gyártott bináris kódot: a komponensek dinamikusan kapcsolt könyvtárait (*dynamic linked library*) vagy a

DLL-eket. Akkor tulajdonképpen mi is az ASP? Az ASP egy technológia, mely lehetővé teszi dinamikus és interaktív weblapokból álló webalkalmazások előállítását.

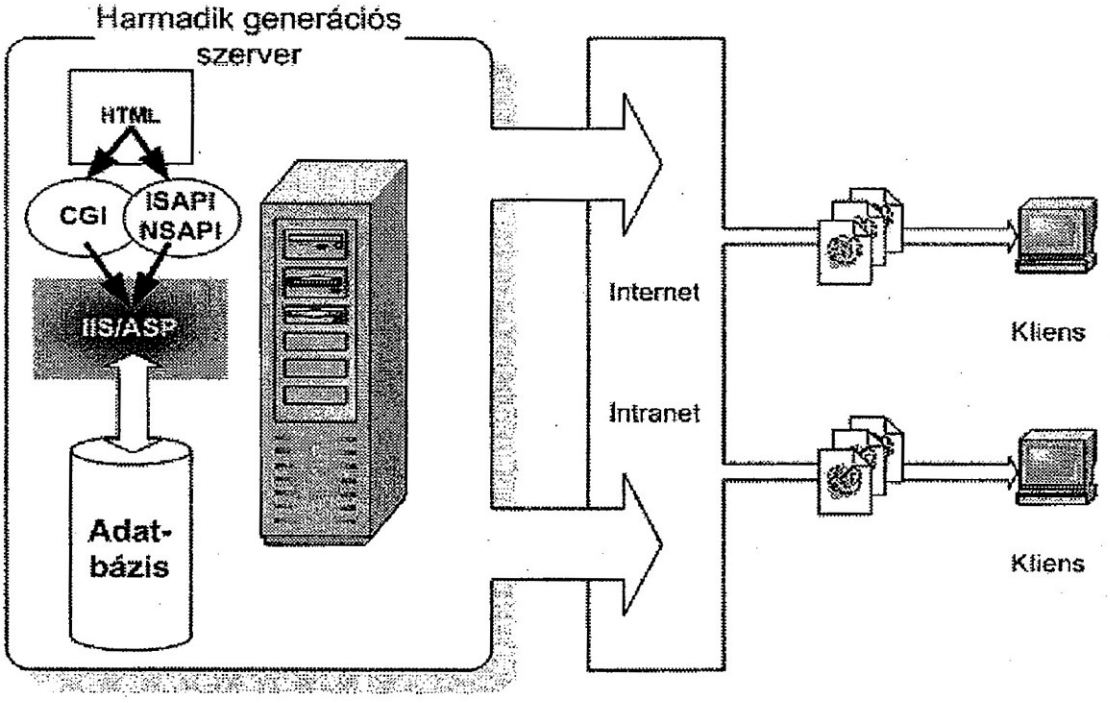
Az ASP, HTML kód interpretálása mellett, módot nyújt arra, hogy úgynevezett ASP objektumokat hozzunk létre, melyek segítségével szerveroldalon a legkülönbözőbb feladatokat is elvégezhetjük, mielőtt egy új HTML lapot létrehozunk és a szerver közvetítésével a kliensnek elküldjük. Ez a technológia fontos előnyökkel rendelkezik a hagyományos szerverekkel szemben.

Legelőször, vegyük észre, hogy a szerveroldali szkriptek nem függnek a kliens platformjától. A szervertől visszaküldött információ "sima" HTML kód vagy ennek későbbi származékai, így bármelyik böngésző értelmezheti.

A második fontos újdonság, amivel az APS jelentkezik abban áll, hogy egy sor előregyártott vagy belső objektummal rendelkezik ugyanannyi feladat megoldására. Így például az ASP olyan objektumokkal rendelkezik, melyek megkönnyítik a kliens adatainak lekérdezését, illetve a szerver válaszának elküldését, lehetővé teszik meghatározott szintű változók tárolását és kezelését és sok más fontos szerveroldali aktivitást. Az ASP belső objektumaival a következő fejezetben fogunk megismerkedni.

A harmadik lényeges újítás az ASP képessége külső objektumokkal kommunikálni. Ezeket a külső objektumokat nevezzük ASP komponenseknek. Az ASP komponensekkel az 5. fejezetben foglalkozunk.

Az 1.3. ábra bemutatja az ASP szerverek felépítését.



1.3. ábra Harmadik generációs szerver

Mint látjuk, az ASP szerver az ISAPI/NSAPI rutinokat is használja, hogy kapcsolatot teremtsen az webszolgáltatásokat nyújtó IIS-sel (**Internet Information Services**), a tulajdonképpeni webszerverrel. A következő szakaszban közelebbről megvizsgáljuk az **Internet Information Services** (IIS) webszervert.

1.2 Az IIS és a webalkalmazások

Azért, hogy ASP alkalmazásokat előállítsunk, ellenőrizzük és végül közretegyük, több Windows komponensre van szükségünk.

Legelőször is rendelkeznünk kell egy böngészővel. A Windows 2000 vagy XP operációs rendszerrel ellátott gépeken az **Internet Exporer 5.0** vagy későbbi be van építve a rendszerbe. Én a 6.0 verziót használom pillanatnyilag, de ez a különbség nem mérvadó.

Ideális esetben egy webszerverre Windows 2000 Server operációs rendszerrel és egy kliensgépre Windows 2000 Professional vagy XP Professional operációs rendszerrel van szükségünk. A szerver- és a kliensgép természetesen egyazon hálózatban kell legyen és legalább webadminisztrátori jogokkal kell rendelkeznünk a webszerveren. Ilyen esetben az ASP alkalmazások a Windows 2000/XP Professional gépen készülnek és a webszerverre lesznek átültetve. Ilyen körülmények azonban ritkán adottak és a könyvben bemutatott agyag elsajátítására, valamint példaanyagának megtekintésére nem szükséges.

Tanulmányi célokra elegendő, ha egy Windows 2000/XP Professional géppel rendelkezünk, de feltétlenül legyünk adminisztrátori jogokkal felruházva. A továbbiakban ezt adottnak fogom tekinteni, és ebből indulok ki. Ez azt jelenti, hogy a gépünk egyszerre lesz webszerver és kliens is. Nézzük meg, hogy kell berendezni gépünket, hogy ASP webprogramokat előállíthassunk!

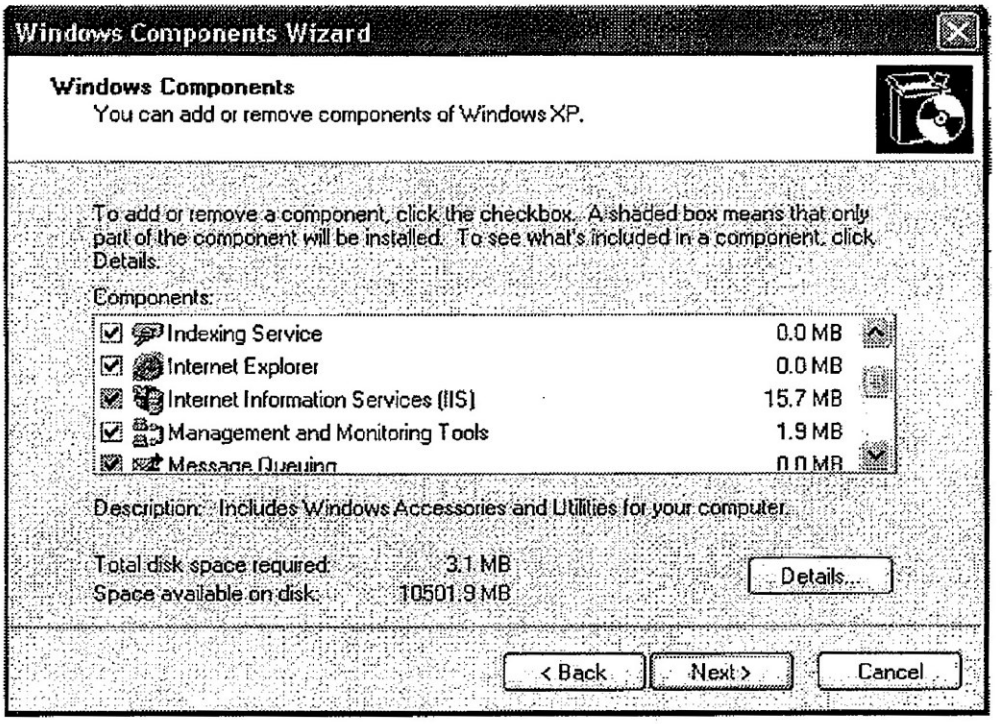
1.2.1 Az Internet Information Services telepítése és beállítása

Az **Internet Information Services 5.x** (IIS) a Windows 2000 Professional webszerver szoftvere. Ha nincs a gépünkre telepítve, akkor ez legyen az első gondunk. Ugyanis az IIS telepítésével az ASP 3.0 is a gépünkre lesz telepítve.

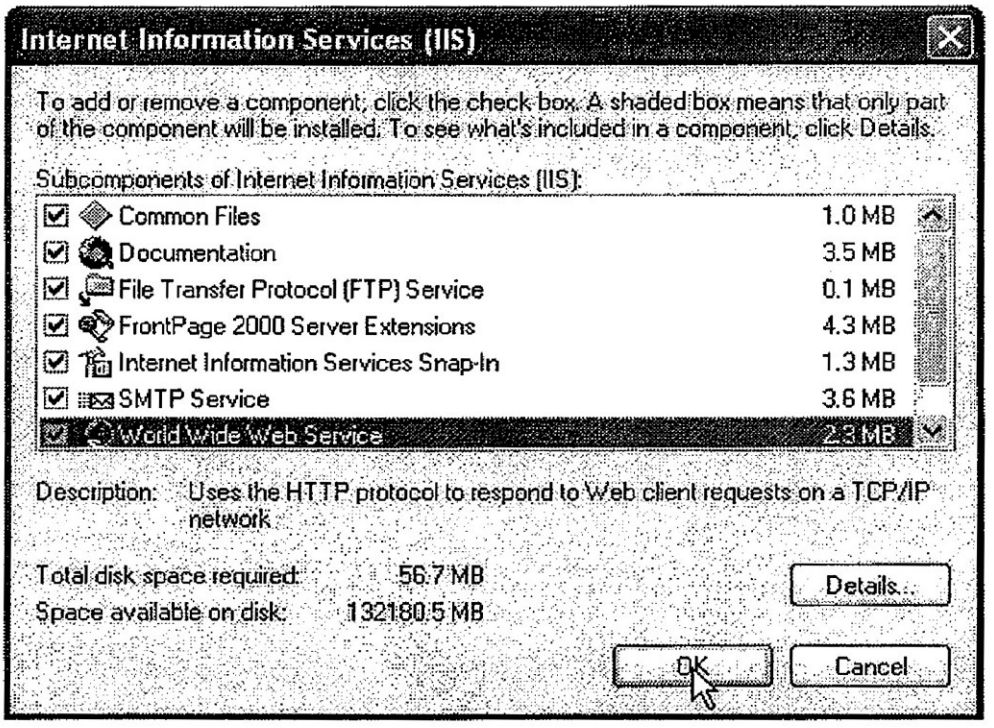
A **Start|Settings|Control Panel** menüparanccsal nyissuk ki a **Control Panel** ablakot és kattintsunk az **Add/Remove Programs** parancsikonra. A megjelenő ablak bal oldali menüsávjában kattintsunk az **Add/Remove Windows Components** menüparancsra, hogy elindítsuk a **Windows Components Wizard** varázslót. A varázsló ablakában a Windows komponensekből álló listát látunk. Minden bejegyzés előtt egy

jelölőnégyzet (*check box*) van. Ha a jelölőnégyzet „ki van pipálva", akkor a komponens már a gépünkön van, ellenkező esetben a jelölőnégyzet üres.

Ha az IIS még nincs a gépünkre telepítve, „pipáljuk ki" a megfelelő jelölőnégyzetet és kattintsunk a **Details...** parancsgombra, hogy kiválasszuk az IIS komponenseit (lásd az 1.4. ábrát).



1.4. ábra A Windows Components Wizard ablaka



1.5. ábra Az IIS komponensei

Az IIS komponenseit felsoroló listában „pipáljunk ki" minden komponenst (lásd az 1.5. ábrát) és az **OK** parancsgombbal térjünk vissza az előbbi ablakra, ahol a **Next>** parancsgomb megnyomásával elindíthatjuk az IIS telepítését. A telepítés folyamán szükségünk lesz a Windows 2000 Professional, illetve a Windows XP Professional CD lemezére.

Nézzük meg, mi került a gépünkre?

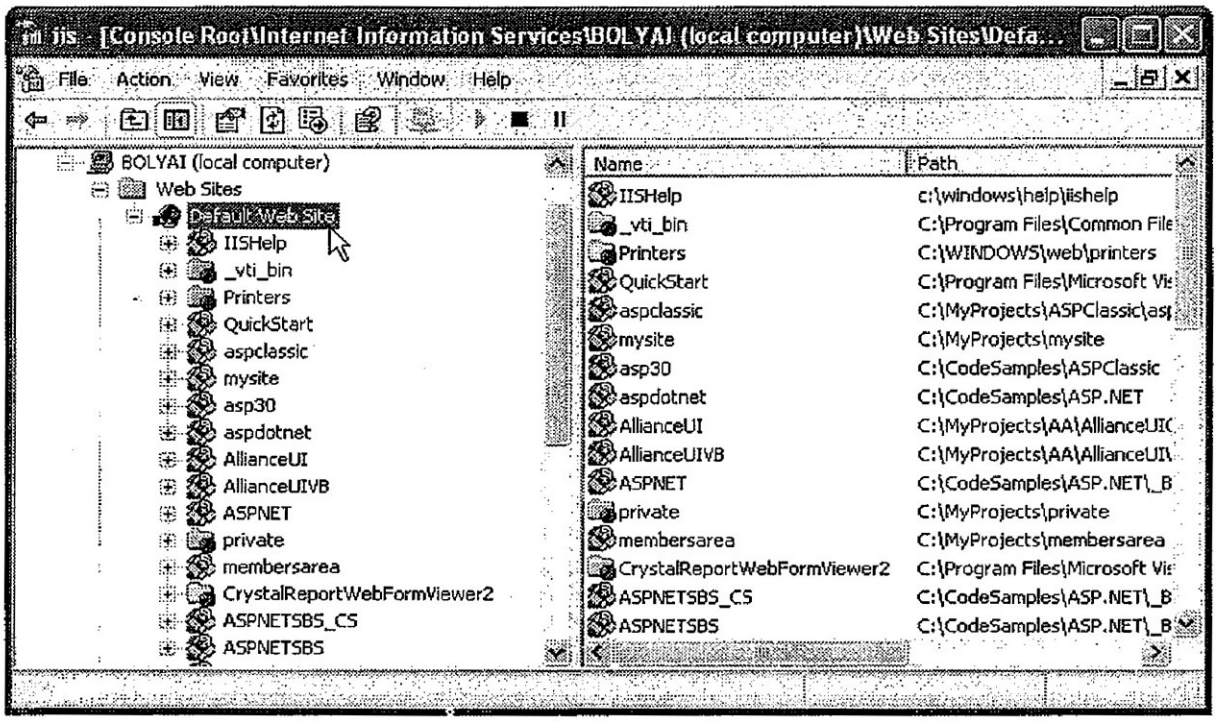
Az IIS telepítésével új mappák jöttek létre. Az IIS a `\winnt\system32\inetsrv` mappába került és létrejött az `\InetPub` mappa több almappával. Az `\InetPub\AdminScripts` mappa egy pár VBScript szkriptet tartalmaz, amit a webszerver kezelésére használhatunk fel, mint például a különféle szolgáltatások elindítására, szüneteltetésére vagy megállítására. Az `\InetPub\Scripts` mappa üres és arra szolgál, hogy ide helyezzük el saját szkriptjeinket. Az `\InetPub\iissamples\home` mappa ASP példalapokat tartalmaz, míg az `\InetPub\iissamples\sdk` VBScript és JScript szkriptekkel szemlélteti az ASP nyújtotta egyes lehetőséget. Az `\InetPub\webpub` mappa is üres. Ide kerülnek a **Publish** varázslóval publikált weblapok. Végül a telepített szolgáltatásoktól függően, több root végződésű mappát látunk: `\InetPub\ftproot`, `\InetPub\mailroot`, `\InetPub\nntproot` és `\InetPub\wwwroot`. Ezek a mappák a megfelelő szolgáltatások (sorrendben az FTP, az Email, az News és a Web szolgáltatások) alapértelmezett mappái. Minket az `\InetPub\wwwroot` mappa érdekel most. Ez a mappa a webszerverünk által létrehozott webhely törzsmappája. Rendeltetése szerint a webalkalmazásokat meghatározó alapmappákat kellene tartalmaznia, amelyekről hamarosan szó esik. A gyakorlatban viszont kihasználjuk az IIS-nek azt a képességét, hogy bármelyik mappában elhelyezett webalkalmazást felismer és nem fogjuk feltétlenül az alapmappákat az `\InetPub\wwwroot` mappába elhelyezni.

Természetesen egy modern operációs rendszertől elvárhatjuk, hogy a kínált szolgáltatásokkal együtt, kényelmes felhasználói felületet is rendelkezésünkre bocsásson. Így van ez az IIS-nél is. A Windows 2000/XP Professional a **Microsoft Management Console** (MMC) alkalmazást kínálja a szolgáltatások kezelésére. Az IIS elérésére legalább kétféleképpen juthatunk el az MMC-hez, vagy a konzolhoz.

Ha Windows 2000 Server-rel dolgozunk, induljunk el a **Start** menüből, válasszuk ki az **Administrative Tools** menüparancsot, majd az **Internet Services Manager** (ISM) menüparancsot! Ha Windows 2000 Professional-lal dolgozunk, a **Start** menün, a **Control Panel** menüparancson és az **Administrative Tools** parancsikonon keresztül eljutunk az **Internet Services Manager** (ISM) parancsikonjához. A menedzser-ablakban láthatjuk az **Internet Information Services** (IIS) ablakát. Az **Internet Services Manager** parancsikont sokszor fogjuk használni, ezért tanácsos a képernyőre másolni.

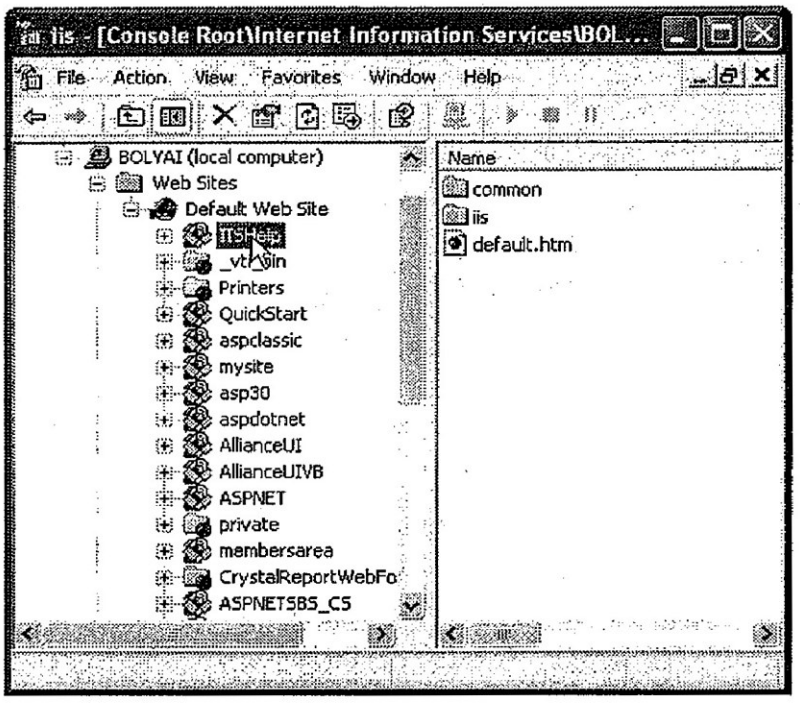
Egy másik lehetőség, hogy az IIS-hez eljussunk egy kis kitérővel jár, de talán még többet is nyújt. Az ISM a **Microsoft Managment Console**-ban (MMC) fut, amit a **Start** menü **Run** menüparancsával mmc beadással is indíthatunk. A megjelent konzolablak **Console** menüjében válasszuk ki az **Add/Remove Snap-In** menüparancsot. Az ezután megjelenő párbeszédablakon kattintsunk az **Add** parancsgombra és a következő ablak listájából válasszuk ki az **Internet Information Services** bejegyzést. Kattintsuk az utolsó ablak **Add**, majd a **Close** parancsgombjára. Az **Add/Remove Snap-In** című ablakon nyomjuk meg az **OK** parancsgombot. Ezzel elkészítettük az MMC konzolt az IIS kezelésére. Állítsuk be az ablakot tetszés szerint és válasszuk ki a **Console** menü **Save As...** menüparancsát. Mentsük a konzolt az elvégzett beállításokkal az IIS.mmc állományba a képernyőre. Innen bármikor könnyen meghívhatjuk, ha az állomány ikonjára kétszer rákattintunk.

Most nyissuk ki az IIS ablakát az **Internet Services Manager** vagy az IIS.mmc parancsikonnal, amit a képernyőre helyeztünk. Az IIS ablaknak hasonlónak kell lenni az 1.6. ábrán látható ablakhoz.



1.6. ábra Az IIS ablaka az MMC konzolban

Az IIS ablak hasonlít a Windows Intézőhöz (*Windows Explorer*). Az ablak bal részén egy fanézetet (*tree view*) látunk. Legfelül az IIS van, alatta a szerver, melynek neve ebben az esetben **home*, és alatta a különféle szolgáltatások: az alapértelmezett FTP-hely (*Default FTP Site*), az alapértelmezett webhely (*Default Web Site*) és az alapértelmezett virtuális SMTP szerver (*Default Virtual SMTP Server*) és (ha szervergépen dolgozunk) az alapértelmezett virtuális NNTP szerver (*Default Virtual NNTP Server*). Minket most az alapértelmezett webhely érdekel.



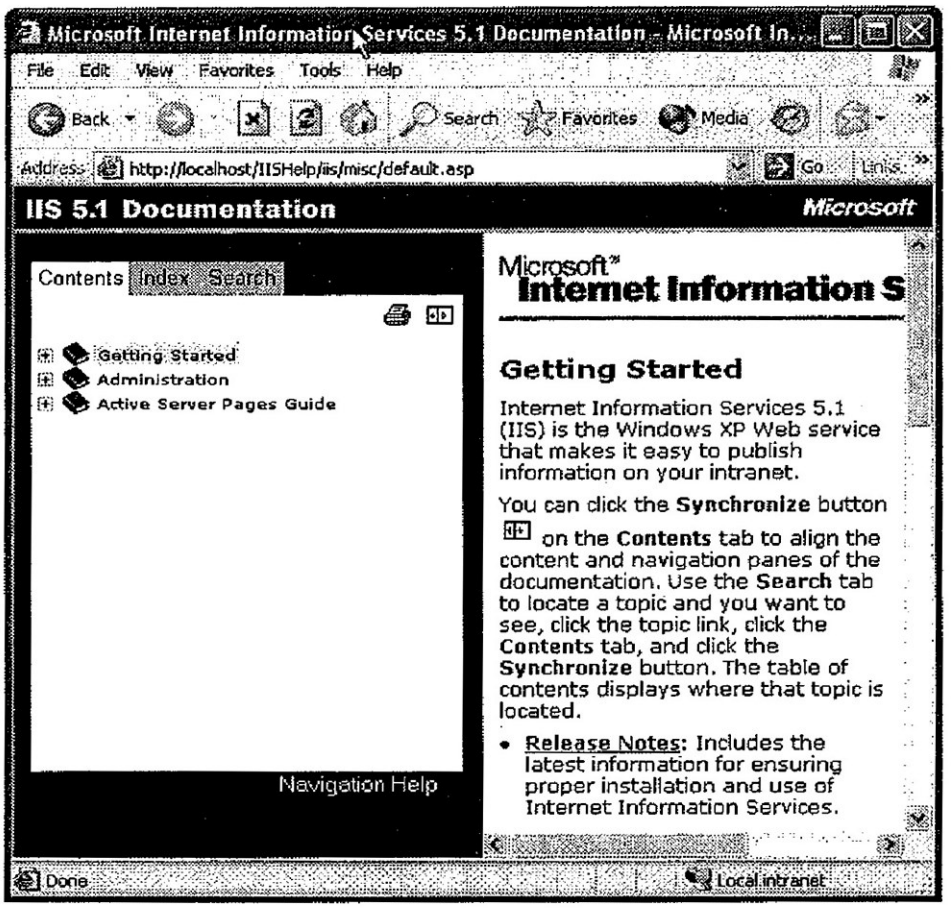
1.7. ábra Az IISHelp webalkalmazás az IIS ablakban

Az alapértelmezett webhely alatt több mappához hasonló ikont, glóbusszal vagy glóbusz nélkül, és több nyitott dobozban levő glóbuszt ábrázoló ikont látunk. A mappákat glóbusszal ábrázoló ikonok virtuális mappáknak felelnek meg, a dobozt ábrázoló ikonok virtuális alkalmazásokat jelenteket, míg glóbusz nélküli mappák közönséges mappákat jelképeznek. Ezeket a fogalmakat nemsokára tisztázni fogjuk. Egyelőre nézünk egy kicsit körül a webhelyünkön!

Az IIS ablakban kattintsunk az IISHelp ikonra és nézzük meg az ablak jobboldalán felsorolt elemeket (1.7. ábra)! Az ablak baloldalán az IISHelp alatt csak mappákat látunk, az ablak jobboldala a kijelölt mappák tartalmát is megmutatja, éppen úgy, mint a Windows Intéző. Most kattintsunk az egér jobb gombjával az IISHelp ikonra és a megjelenő környezetmenüből (*context menu*) válasszuk ki az **Open in Browser** menüparancsot.

Ezzel az IISHelp webalkalmazás kezdőlapja megjelenik a böngészőben (1.8. ábra). Ez a webalkalmazás nem más, mint az IIS súgója és sok mindenben segítségünkre lehet. Most azonban a megjelenített lap címe érdekel minket:

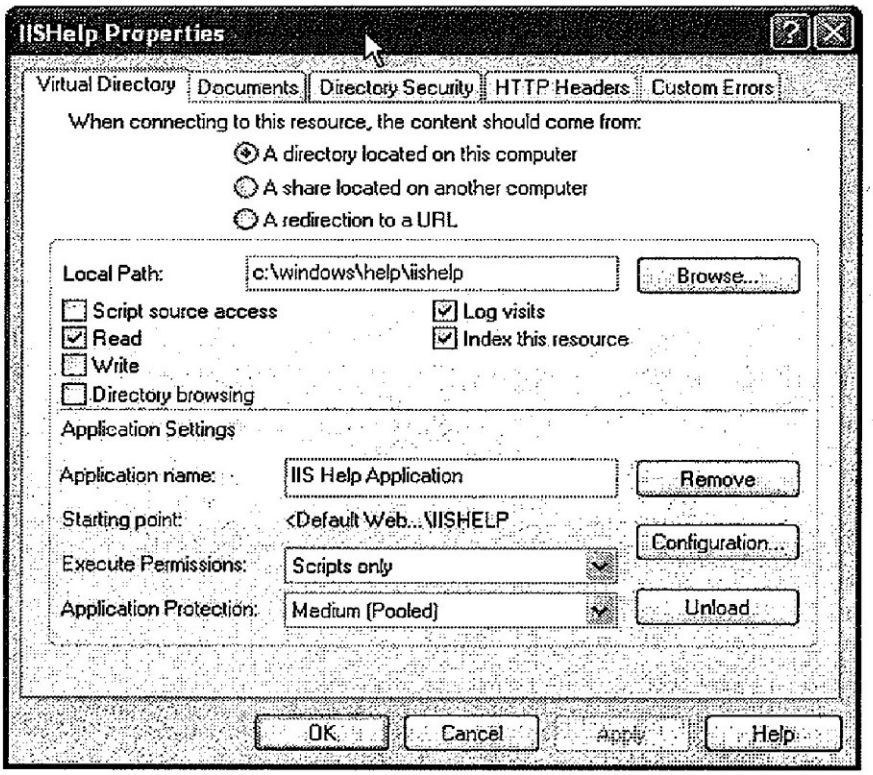
```
http://localhost/IISHelp/iis/misc/default.asp.
```



1.8. ábra Az IISHelp webalkalmazás a böngészőben

Legelőször az tűnik fel, hogy az IIS ablakban az IISHelp webalkalmazásnál a `default.htm` lap volt megadva, most viszont a `default.asp` lap van a címben feltüntetve. Ha a böngészőbe a `http://localhost/IISHelp/default.htm` lapot kérjük, az eredmény ugyanaz. Próbáljuk az eredeti `default.htm` állományt megjeleníteni, vagy legalább a forráskódját megtekinteni.

Ezért kattintsunk még egyszer az egér jobb gombjával az ISSHelp ikonra és e környezetmenüből válasszuk ki ezúttal a **Properties** menüparancsot. A megjelenő párbeszédablak **Virtual Directory** lapján a **Local Path** címke alatt (1.9.ábra) megtaláljuk az állomány elérési útját: `\winnt\help\iishelp`. Nézzük meg ennek a mappának a tartalmát! A mappában ténylegesen megtaláljuk az ismert almappákat és a keresett `default.htm` állományt. Kattintsunk most kétszer erre az állományra, hogy megjelenítsük a böngészőnkben! Ahelyett, hogy megmutatná nekünk a HTML lapot a böngésző a `\winnt\Help\iisHelp\iis\misc\default.asp` állományt akarja letölteni. Ha a böngészőtől a `\winnt\Help\iisHelp\default.htm` lapot kérjük, az eredmény nem változik. Mi lehet ennek a magyarázata?



1.9. ábra Az IISHelp virtuális alkalmazás tulajdonságai

Ne töltsük le az állományt, hanem nézzük meg a `default.htm` állomány forráskódját:

```
<HTML>
  <HEAD>
  <TITLE>
  </TITLE>
  <META HTTP-EQUIV="Refresh" CONTENT="0; URL=iis/misc/default.asp">
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

Most már világos, hogy a `default.htm` állomány nem jelenít meg semmit, csupán a már ismert `\winnt\Help\iisHelp\iis\misc\default.asp` állományra átirányítja a böngészőnket. Ezt az állományt az `.asp` kiterjesztése miatt a böngésző nem ismeri fel és felajánlja a letöltését. Az `.asp` kiterjesztésű állomány nem egy HTML állomány, hanem egy ASP állomány és a böngésző nem tudja értelmezni.

Miért láttuk azonban az alkalmazást legelőször a böngészőben és miért nem a második próbálkozáskor? A két eset között egy lényeges különbség áll fenn.

Amikor a böngészővel a `http://localhost/IISHelp/default.htm` lapot kértük, akkor a kérést az IIS-nek, a webszervernek intéztük. A webszerver megtalálta a kért lapot, értelmezte, felismerte az átirányítást, megtalálta az átirányítás céllapját, a `http://localhost/IISHelp/iis/misc/default.asp` lapot felismerte, hogy

egy ASP lapról van szó és továbbította ezt az ASP lapot az ASP-nek. Az ASP végigolvasta a lapot, elvégezte a talált utasításokat, majd ezek alapján létrehozott egy HTML lapot a memóriában, amit visszaküldött a böngészőnknek. Végül a böngésző a szervertől kapott „sima" HTML kódot megjelenítette.

A második alkalomkor, amikor a `\winnt\Help\iisHelp\default.htm` állományt akartuk a böngészővel megjeleníteni, a kérést nem a webszervernek intéztük, hanem az operációs rendszerünknek. A rendszer megtalálta a kért lapot és kódját visszaküldte a böngészőnek. A böngésző végrehajtotta a lap kódját, azaz egy újabb lapot, a `\winnt\Help\iisHelp\iis\misc\default.asp` lapot kérte a rendszertől. A rendszer ezt a lapot is megtalálta és kódját változatlanul visszaküldte a böngészőnek. A kapott kóddal, mivelhogy nem HTML kód, a böngésző nem tud mit kezdeni és így, a beállításoknak megfelelően, felajánlja letöltésre.

Ez a rövid kísérletezés, gondolom rávilágított az ASP szerepére. A továbbiakban közelebbről fogjuk megvizsgálni az ASP működését.

1.2.3 Webalkalmazások és munkamenetek

A hagyományos programokkal megszokott fejlesztő felteheti a kérdést: „Lehetséges ilyen kevés eszközzel komoly webalkalmazásokat létrehozni?" Ha rágondolunk, mi mindenre van szüksége egy programozónak, mikor egy C/C++ vagy egy Visual Basic programot elkészít, akkor tényleg jogos a kérdés. Lokális és globális változókra, több állományon átterjedő programmodulokra és a köztük lebonyolítandó biztonságos adatcserére van szükségünk ahhoz, hogy egy tényleg hasznos és jól működő programot előállítsunk. Hogy tudjuk ugyanezt egy webalkalmazással elérni, ahol ráadásul több felhasználó, kliens futtatja ugyanabban az időben ugyanazt az alkalmazást?

ASP válasza ezekre a kérdésekre egyértelmű: igen, mindez lehetséges. Hogy ezt megértsük, új fogalmakkal kell megismerkednünk.

Egy webalkalmazás egy sor weblapból (mint például a HTML vagy az ASP lapok), a hozzátartozó erőforrásokból (mint például a kép- és hangállományok) és egy sor külső komponensből és a hozzátartozó speciális állományokból (mint például ADO és az adatbázisok) áll, azzal a céllal, hogy egy bizonyos feladatra megoldást kínáljon.

Ahhoz, hogy egy webalkalmazás működőképes legyen, a webszerver amelyen fut, képes kell legyen az alkalmazásra és kliensre jellemző adatok strukturált tárolására, még akkor is, ha a klienssel történő adatcsere egy időre megszűnik. Hogy jobban megérthessük miről van szó, képzeljünk el egy konkrét esetet: egy sporttermékeket gyártó cég elindítja internetes vásárlóközpontját és mi vagyunk megbízva a szükséges webalkalmazás elkészítésével. Hogy kell működjön egy ilyen webalkalmazás?

Mindjárt az elején világos, hogy az adatokat osztályoznunk kell. Kezelnünk kell olyan adatot, mely az egész alkalmazás folyamán jelen kell legyen, mint például a látogatók száma. Ez nem csak a látogatók számára érdekes. A cég legalább annyi hasznát veszi az üzleti stratégiájának eldöntésében. Milyen időszakban, milyen időszakként, hányan látogatták melyik lapot és mit és mennyit rendeltek meg? Kell-e a raktárállományt felújítani, van-e a kínált termékek közül olyan, melynek a kínálása nem hozza meg a várt eredményt? Ilyen és hasonló adatok azután is meg kell maradjanak, miután egy látogató elhagyta az alkalmazást és más honlapra tért át. Azonkívül az adatközlésnek dinamikusnak kell lenni a látogató felé. Ha az alkalmazás folyamán a raktárból egy bizonyos termék elfogyott, mert más látogatók felvásárolták, ezt a még aktív vásárlókkal azonnal közölni kell.

Az adatok egy másik csoportja viszont inkább egy adott látogatóval kapcsolatos és nem érdekli más látogató, vagy éppen nem kell, hogy más látogató is tudomást szerezzen róluk. Ebbe a csoportba tartozik például a látogatók webprofilja. Minden látogatónak meg kell adni a lehetőséget, hogy a weblapot saját igénye szerint átalakíthassa. Legközelebbi látogatása alkalmával a kialakított egyedi profilját, a már megszokott lapformát szeretné viszontlátni. Ide sorolhatók a látogatók személyi adatai is: mit, mikor, mennyit vásárolt vagy rendelt meg, valamint a látogatót azonosító adatok A látogatók adatait tanácsos csak egyszer, az első látogatás alkalmával lekérdezni. Legközelebb csak az azonosítónevet és a jelszót kérjük tőle. Mindemellett ezeket az adatokat több weblapon át kell biztonságosan végigvinnünk.

Egy pillanatig sem szabad megfeledkeznünk egy lényeges különbségről, ami a hagyományos, önálló (*stand-alone*) alkalmazások és a webalkalmazások között fennáll: egy webalkalmazásnak egyazon időben több, akár több száz felhasználója is lehet. Már ez a tény egymagában elég nyomós ok arra, hogy az adatok kezelése a legnagyobb körültekintéssel történjen.

E rövid áttekintésből is láthatjuk milyen fontos az adatok strukturált kezelése. Most ne gondoljunk a feladat megoldására feltétlenül szükséges adatbázisra és ne foglalkozzunk az adatbázis felé és felől történő adatcserére. Ezzel később fogunk foglalkozni.

Mindezekre a kérdésekre az ASP megoldása két új fogalom bevezetésén alapszik. A virtuális alkalmazásról (*virtual application*) és a munkamenetről (*session*) van szó.

Mint láttuk, egy webalkalmazás egy sor összetartozó weblapból, erőforrásból és esetleg komponensekből áll. A webalkalmazás ezen részei, természetesen különböző mappákban kapnak helyet a webszerveren, de végül is mind egy átfogó mappában, az alapmappában vannak. Ideális, ha ebben az alapmappában nincsenek a webalkalmazástól idegen állományok. Ez megkönnyíti az alkalmazás karbantartását és növeli az áttekinthetőségét. A webalkalmazást átfogó alapmappát a webszerveren definiálnunk kell. Ezzel egy virtuális alkalmazást hozunk létre. Röviden: egy virtuális alkalmazás

egy webszerveren definiált webalkalmazás, vagy másképpen kifejezve, a webszerver memóriájába töltött webalkalmazás. Nagyon sokszor a virtuális alkalmazás és a webalkalmazás fogalmakat felváltva használjuk, mivelhogy egy webalkalmazásnak nincs semmi értelme, ha nincs egy webszerveren definiálva. Én is felváltva fogom a két fogalmat használni. Nemsokára látni fogjuk, miként lehet egy virtuális alkalmazást létrehozni. Egyelőre nézzük meg közelebbről a már létező virtuális alkalmazásokat.

Az IIS telepítésével több virtuális alkalmazás is létrejött. Az egyiket közülük, az IISHelp alkalmazást már ki is próbáltuk. A megfelelő webalkalmazást átfogó alapmappáját is megtaláltuk: `\winnt\help\iishelp`. Ez és a többi alapértelmezett virtuális alkalmazás egy legfelsőbb szintű virtuális alkalmazáson belül, az úgynevezett gyökéralkalmazáson belül fut.

Egy webalkalmazás, miután a szerveren definiáltuk, máris aktív (azaz jelen van a szerver memóriájában), még akkor is, ha egyetlen egy ügyfélgép sem kér egy lapot sem az alkalmazásból. Az alkalmazás aktív marad, ameddig le nem állítjuk vagy ameddig a szervert le nem állítjuk. Ez az egyik lényeges különbség a hagyományos és a webalkalmazások között: egy hagyományos alkalmazást egy felhasználó (ami természetesen egy másik program is lehet) indít el, míg egy webalkalmazást a webszerver indítja el. A webalkalmazás nincs felhasználóhoz, ügyfélgéphez kötve, és amint ezt később látni fogjuk, minden klienstől függetlenül, úgynevezett alkalmazásszintű változókat vagy alkalmazás-változókat (*application-level variable*) képes létrehozni és kezelni. Az alkalmazás-változók az alkalmazás bármelyik lapjáról addig érhetők el, ameddig az alkalmazás aktív.

Eltekintve a különleges esetektől, amiről később még szó esik, mihelyt egy ügyfélgép az alkalmazás egyik lapját kéri a webszervertől, a szerver egy munkamenetet nyit számára. A webszerver munkamenetek segítségével feljegyzést készít a webalkalmazások látogatóiról. A munkamenetek a kliensekhez vannak kötve. Egy munkamenet akkor kezdődik, amikor egy kliens az alkalmazás egyik lapját lekéri és akkor ér véget, amikor a látogató elhagyja az alkalmazást. Minden látogatóhoz külön munkamenetet rendel a webszerver. Egy webalkalmazás a munkamenet segítségével meg tudja különböztetni a látogatókat, és amint ezt később látni fogjuk, munkamenetszintű vagy munkamenet-változókat (*session-level variable*) képes létrehozni és kezelni. A munkamenet-változók az alkalmazás bármelyik lapjáról és csak a munkamenethez hozzárendelt kliens által addig érhetők el, ameddig a munkamenet be nem fejeződik.

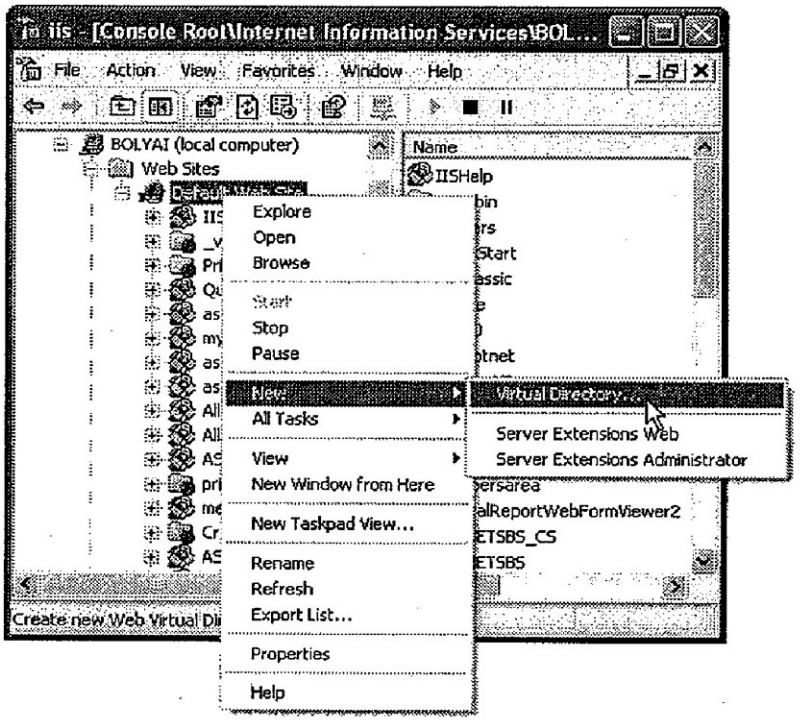
Az alkalmazásszintű és munkamenetszintű változókon kívül a webalkalmazások lapszintű változókat (*page-level variable*) is használnak. Ezek a változók csak azon a lapon érhetők el, amelyiken definiálva vannak.

1.2.4 Virtuális mappák és alkalmazások létrehozása

Az alapértelmezett virtuális alkalmazások mellett mi is létrehozhatunk virtuális alkalmazásokat. A továbbiakban egy virtuális alkalmazást, a könyvben szereplő összes példalapokat tartalmazó alkalmazást, és ezen belül két virtuális alkalmazást hozunk létre: a könyvben szereplő VBScript, illetve JScript szkriptnyelveket használó ASP példalapokból álló alkalmazást.

Abból indulok ki, hogy a kedves Olvasó a letöltött ZIP állományból a rendszerébe másolta az `ASPClassic` mappát. Ebben a mappában, más almappák mellett, két almappa van: `JS` és `VBS`. Ez a két almappa a könyvben szereplő JScript, illetve VBScript példákat tartalmazza. Ezt a három mappát kell az webszervernek, pontosabban a webszerver IIS Internet szolgáltatásának bejelentenünk. Ez úgy történik, hogy megadjuk a mappa fizikai elérési útját és a virtuális elérési útját, azaz az általános erőforráshely meghatározóját, röviden az URL-jét (*Unified Resource Locator*).

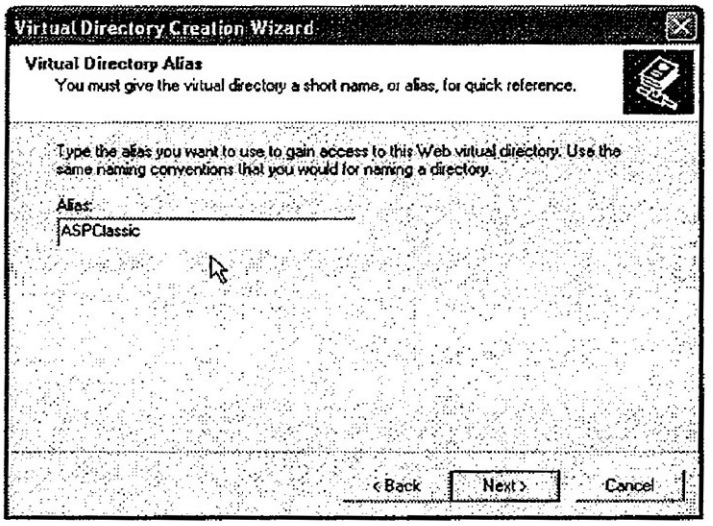
Ha a kedves Olvasó a letöltött ZIP állományban található `ASPClassic` mappát még nem másolta le a gépére, akkor most itt van a legjobb alkalom. A mappát bemásolhatjuk az `\InetPub\wwwroot` mappába vagy bármely más elérhető helyre a gépünkön. Miután ez megtörtént, nyissuk ki az MMC konzolban az IIS ablakát és kattintsunk az egér jobb gombjával az alapértelmezett **Default Web Site** webhely ikonjára. A megjelenő menüben válasszuk ki a **New**, majd a **Virtual Directory** menüparancsot (1.10. ábra).



1.10. ábra Elindítjuk a Virtual Directory Creation varázslót

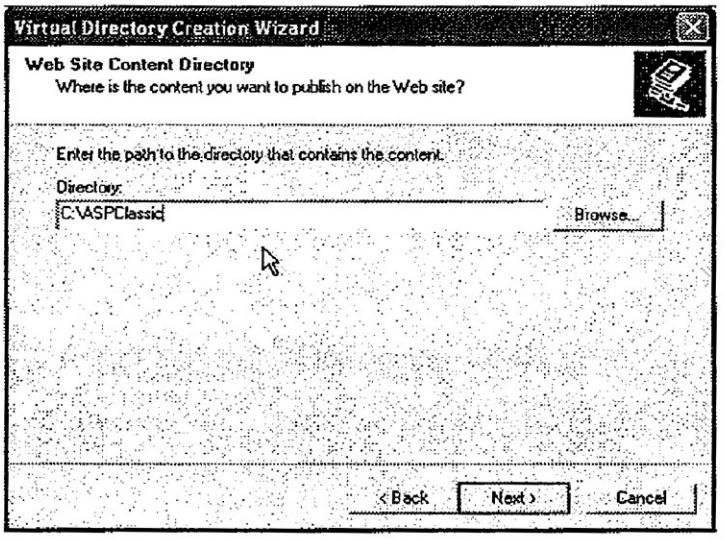
Ezzel elindítottuk a **Virtual Directory Creation** varázslót, mely segíteni fog a virtuális alkalmazások létrehozásánál.

Nyomjuk meg a varázsló kezdőablakán levő **Next >** parancsgombot és a következő ablakon adjuk be az alkalmazás rövidnevét (*alias*): *ASPClassic* (1.11. ábra), majd nyomjuk meg a **Next >** parancsgombot.



1.11. ábra Beadjuk a virtuális alkalmazás rövidnevét

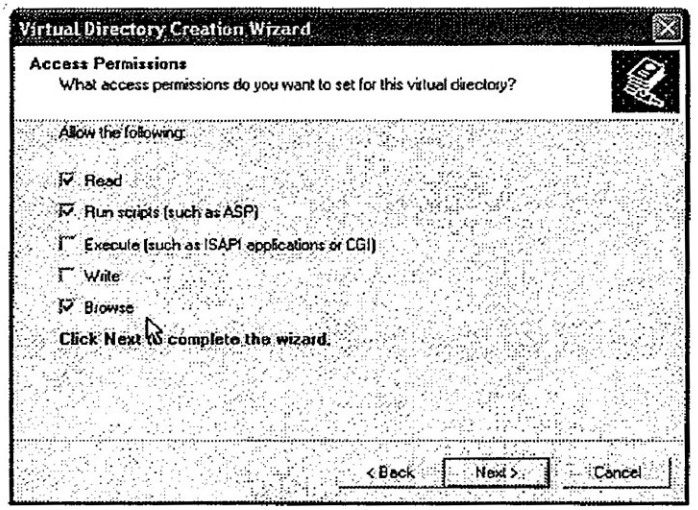
A következő ablakon a **Directory** szövegmezőbe kell megadnunk a webalkalmazás mappáját. Ez az a hely, ahova a letöltött ZIP állományból az `ASPClassic` mappát átmásoltuk (1.12. ábra).



1.12. ábra Beadjuk a webalkalmazás alapmappáját

A **Next >** parancsgombbal átkerülünk az **Access Permissions** ablakra (1.13. ábra), ahol az alkalmazás jogosultságát rögzítjük. Egyelőre elég lesz, ha a mappa tartalma olvasható lesz, ha ASP szkripteket végrehajthatunk, és ha a mappa tartalmát végigböngészhetjük. Ennek megfelelően „pipáljuk ki" a **Read**, a **Run scripts (such as ASP)** és

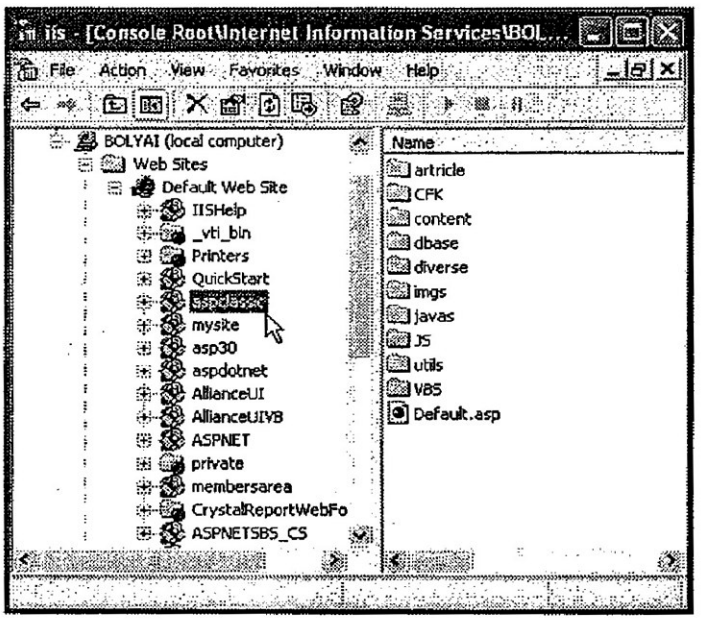
a **Browse** választónégyzeteket és fejezzük be a virtuális alkalmazás létrehozását az ablakon levő **Next >** és a következő ablakon levő **Finish** parancsgombbal.



1.13. ábra Meghatározzuk a virtuális alkalmazás jogosultságát

Ezzel létrehoztunk egy virtuális alkalmazást. Az MMC konzolban az IIS ablaka most az 1.14. ábrán látható ablakhoz hasonló lesz.

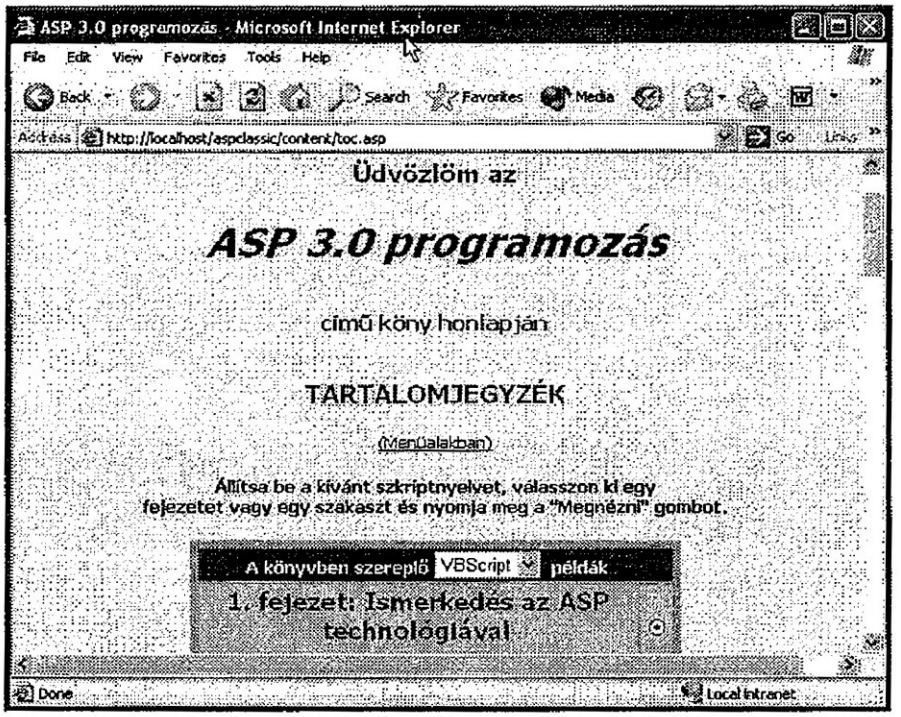
A létrehozott virtuális alkalmazás indítására több lehetőségünk van. Legelőször kattintsunk a jobb egérgombbal az alapértelmezett **Default Web Site** webhelyre és a megjelenő menüben ellenőrizzük, hogy webszolgáltatás tényleg aktív.



1.14. ábra Az ASPClassic virtuális alkalmazás

Ha a webszolgáltatás aktív, akkor a menüben a **Start** menüparancs nem választható ki, a **Stop** menüparancs viszont kiválasztható. Ebben az esetben nem kell semmit tennünk. Ha a webszolgáltatás nem aktív, akkor a menüben a **Start** menüparancs kiválasztható a **Stop** menüparancs viszont nem. Ebben az esetben indítsuk el a webszolgál-

tatást. Most elindíthatjuk az *ASPClassic* alkalmazást, amennyiben a jobb egérgombbal az alkalmazás ikonjára kattintunk és a környezetmenüből kiválasztjuk az **Open in Browser** menüparancsot. Az alkalmazás kezdőlapja megjelenik az **Internet Explorer** ablakában (1.15. ábra).

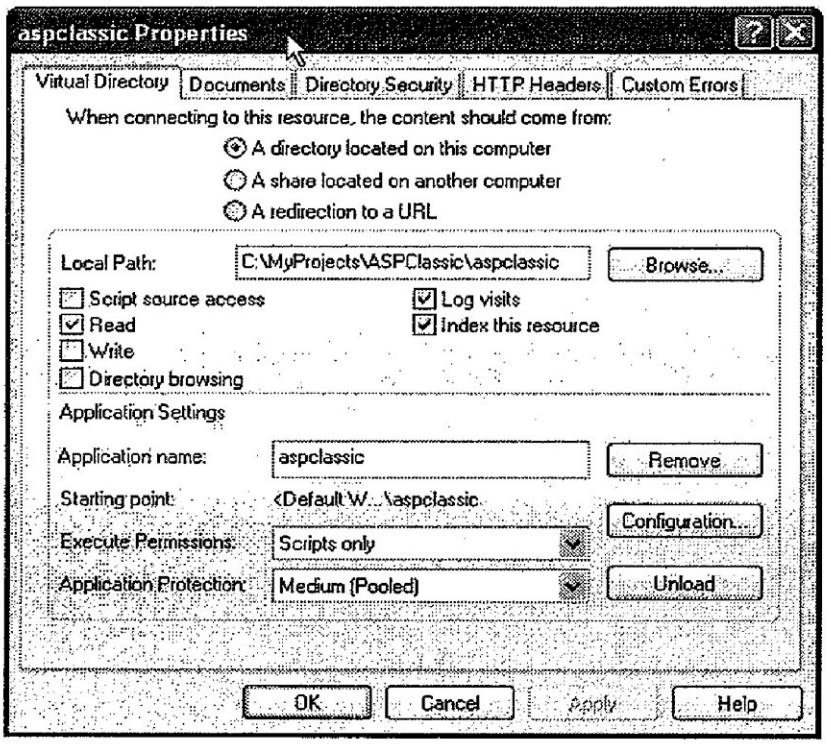


1.15. ábra Az ASPClassic alkalmazás kezdőlapja az Internet Explorer ablakában

Az **Internet Explorer** címsorába láthatjuk az alkalmazás URL címét: `//localhost/ASPClassic`. Ha ezt a címet beadjuk az **Internet Explorer** címsorába, akkor szintén elérhetjük az alkalmazás kezdőlapját.

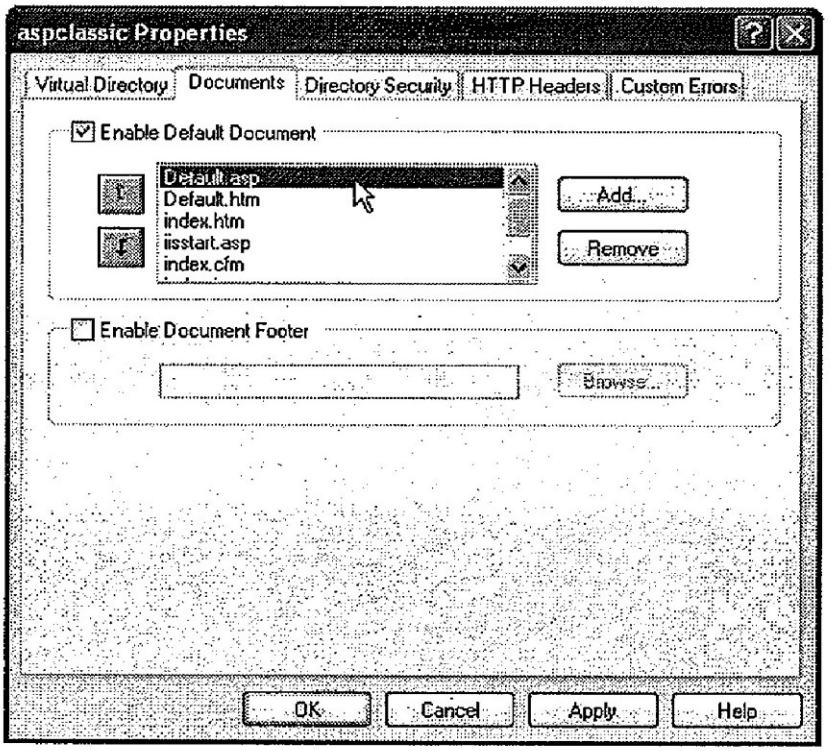
Az IIS ablakban láthatjuk, hogy az ASPClassic virtuális alkalmazás több mappát és a `default.asp` lapot tartalmazza. A `default.asp` lap az alkalmazásunk kezdőlapja és a böngésző ezt a lapot jeleníti meg, ha a címsorba az alkalmazás URL címét beadjuk. Alapbeállításban a virtuális alkalmazások kezdőlapja `default.htm` vagy `default.asp` nevet viselheti, de gyakorlatilag bármilyen nevet választhatunk. Alapvető feltétel, hogy az alkalmazás kezdőlapjának az alapmappában legfelül kell lennie, mert ebben a mappában keresi a webszerver. Ha az alapértelmezett név helyett más nevet akarunk használni, akkor az alkalmazás tulajdonságait megfelelően kell beállítanunk.

Kattintsunk a jobb egérgombbal az *ASPClassic* alkalmazás ikonjára és a megjelenő környezetmenüből válasszuk ki a **Properties** menüparancsot. A megjelenő párbeszédablakon lehetőségünk van az alkalmazás tulajdonságait ismét beállítani vagy módosítani (1.16. ábra).



1.16. ábra Az ASPClassic alkalmazás tulajdonságai

A párbeszéd ablak **Documents** lapján meghatározhatjuk az alkalmazás kezdőlapját (1.17. ábra).

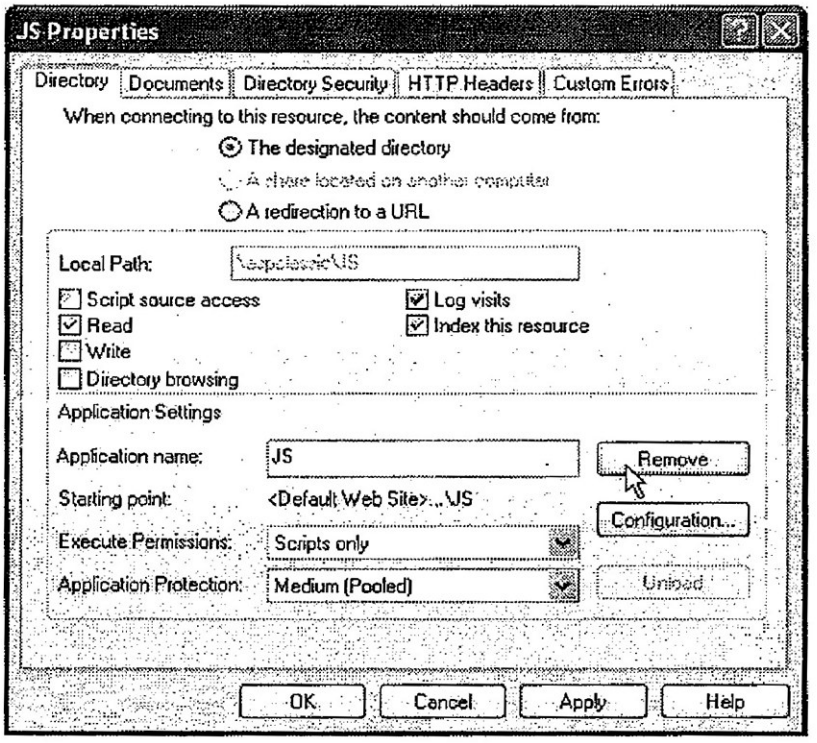


1.17. ábra Az ASPClassic alkalmazás kezdőlapjának meghatározása

Térjünk vissza az IIS ablakára az MMC konzolban (1.14. ábra)! Láttuk, hogy az ASPClassic alkalmazásnak két mappája van: JS és VBS. Mivelhogy ezek a mappák az

ASPClassic alkalmazáshoz tartoznak és az alkalmazást feljogosítottuk, hogy az almappákba böngésszen, ezek az almappák minden ügyfélgépről elérhetők. Ha az **Internet Explorer** címsorába beadjuk az egyik almappa címét, ez például a JS mappa esetében `//localhost/ASPClassic/JS`, akkor a mappa tartalma megjelenik a böngészőben. A későbbiekben azonban szükségünk lesz a JS, illetve a VBS mappában található `global.asa` állományra. A `global.asa` állománnyal és szerepével később részletesen fogunk foglalkozni. Egyelőre annyit jegyezzünk meg, hogy szerepét csak akkor töltheti be, ha egy virtuális alkalmazás alapmappájában van elhelyezve. A mi esetünkben ez nem áll fenn. Ezért átalakítjuk a JS és VBS mappákat virtuális alkalmazásokká.

Kattintsunk a jobb egérgombbal a `JS` mappa ikonjára és a megjelenő menüből válasszuk ki a **Properties** menüparancsot. A `JS` mappa tulajdonságait mutató párbeszédablakon az **Application name** bejegyzés nem aktív. Ha a mellette levő **Create** parancsgombot megnyomjuk, akkor átalakítjuk a mappát egy virtuális alkalmazássá (1.18. ábra). A parancsgomb felirata átváltozik **Remove**-ra és a alkalmazás ugyanazt a nevet kapja, mint a mappa. Ezt az alkalmazásnevet természetesen megváltoztathatjuk, de jelen esetben az alapértelmezett név megfelel nekünk.



1.18. ábra Átalakítjuk a JS mappát virtuális alkalmazássá

A `JS` mappához hasonlóan alakítsuk át a `VBS` mappát is egy virtuális alkalmazássá. A két új alkalmazás a következő címen érhető el: `//localhost/ASPClassic/js`, illetve `//localhost/ASPClassic/vbs`.

Ha ezzel végeztünk, akkor három webalkalmazást hozunk létre:

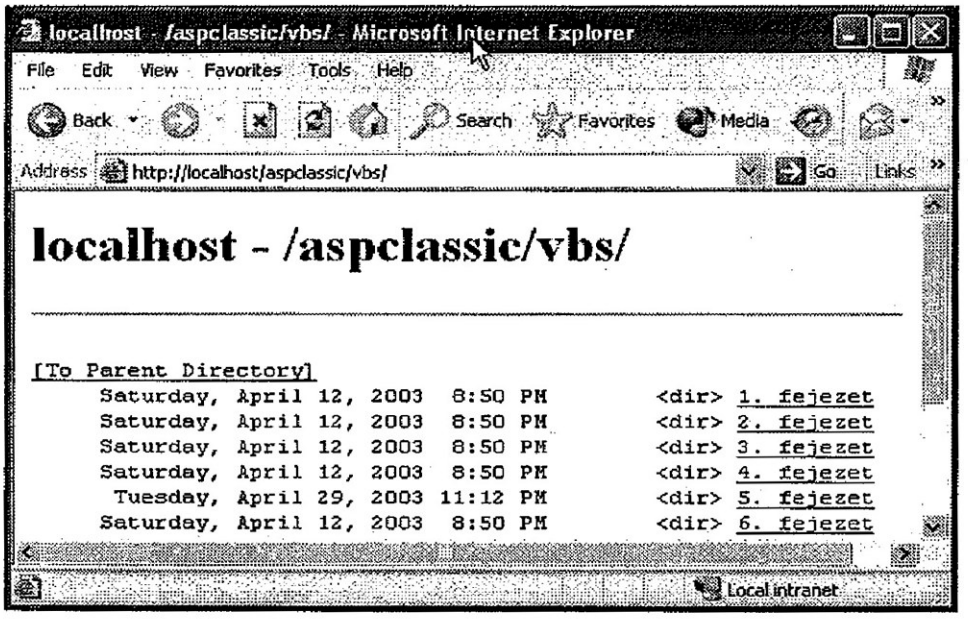
webalkalmazás	cím	tartalom
ASPClassic	`http://localhost/ASPClassic`	összes példák
VBS	`http://localhost/ASPClassic/vbs`	VBScript példák
JS	`http://localhost/ASPClassic/js`	JScript példák

A következő szakaszban megvizsgáljuk, miként hívhatjuk meg, és hogyan kell értelmezzük a példákat.

1.2.5 A könyvben szereplő példákról

Hívjuk le a *VBS* webalkalmazás kezdőlapját! Indítsuk el a böngészőnket és adjuk be az alkalmazás címét: `http://localhost/ASPClassic/vbs`. A böngészőben a `VBS` mappa tartalmát látjuk (1.19. ábra).

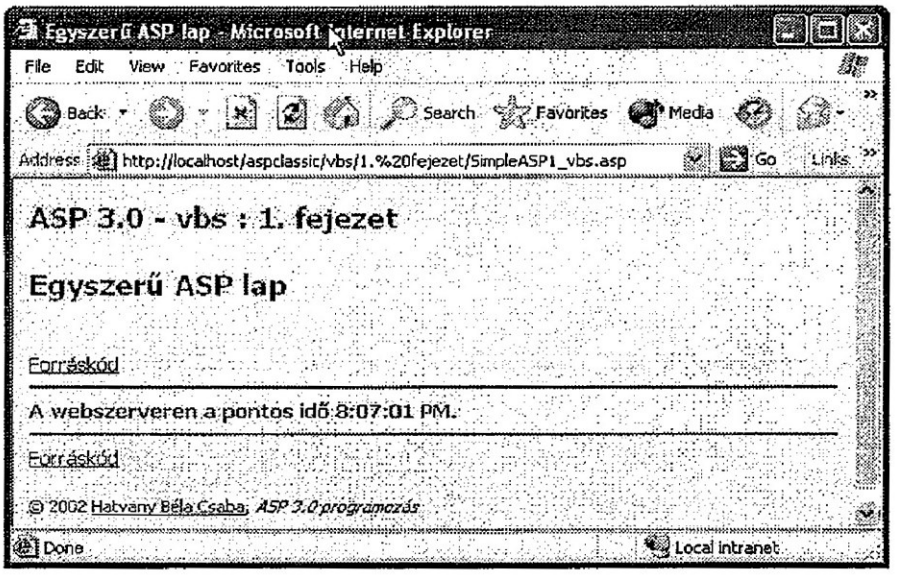
Mivelhogy az alkalmazásnak megengedtük, hogy a mappákban böngésszen és ugyanakkor nem helyeztünk a `VBS` mappába egy állományt, melynek neve megegyezik a tulajdonságoknál megadott kezdőlapok nevével, az alkalmazás kezdőlapként a `VBS` mappa tartalmát mutatja meg. A meghatározott tulajdonságok alapján jogosultak vagyunk tovább böngészni.



1.19. ábra A VBS alkalmazás kezdőlapja

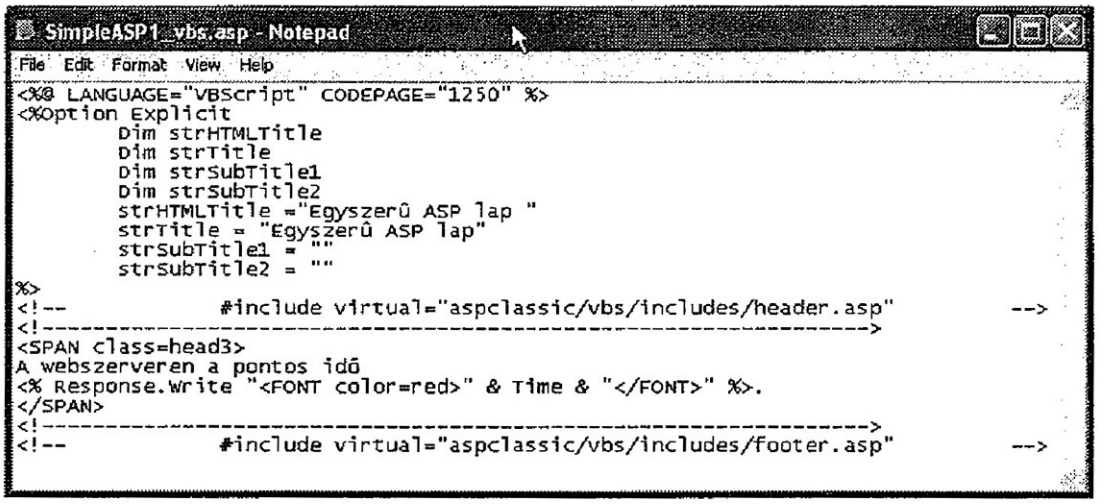
Kattintsunk az 1. fejezet kapcsolatra (*link*) és tekintsük meg a mappa tartalmát, majd hívjuk le az ott talált `SimpleASP1_vbs.asp` ASP lapot (1.20. ábra). Ez egy egyszerű ASP lap. Ugyanúgy, mint a többi példa, három részből áll. A részeket egy vízszintes vonal választja el. A szoros értelemben vett példa a lap második része, tehát a két víz-

szintes vonal között megjelenő rész. Jelen esetben ez az egyszerű ASP lap megmutatja, hogy lehet egy egyszerű szöveget megjeleníteni.



1.20. ábra Egy egyszerű ASP lap a böngészőben

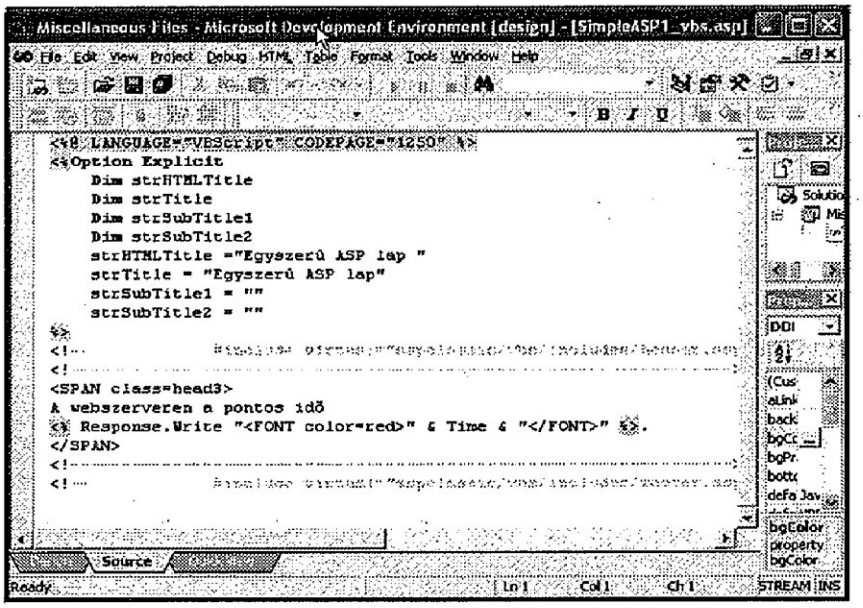
Az első vízszintes vonal feletti rész információt közöl a lapról. Célja az ASP lap könnyebb azonosítása. Így például az első sor arról értesít, hogy a jelenlegi lap VBScript példa és hogy az első fejezethez tartozik. A második (és néha még egy) sor röviden leírja, melyik témakörhöz kapcsolódik a példa. Az első rész utolsó sora a Forráskód kapcsolatot tartalmazza, amiről később lesz szó.



1.21. ábra A SimpleASP1_vbs.asp állomány a Notepad-ben

A `SimpleASP1_vbs.asp` állományt a **Notepad**-ben 1.21. ábrán, az **InterDev** fejlesztőrendszerben pedig a 1.22. ábrán láthatjuk. Elvileg ASP szkripteket a **Notepad**-del is lehet előállítani, de nagyobb alkalmazások programozására nem alkalmas. Szerencsére az Interneten elérhető egy kitűnő programszerkesztő, mely az **InterDev**-hez hasonlóan tetszés szerint beállítható színezéssel kiemeli a szintaxiselemeket. A

Textpad szerkesztőről van szó, amit a `http://www.textpad.com` címről tölthetünk le. Természetesen akármelyik másik ASCII szövegszerkesztőt is használhatunk.



1.22. ábra A SimpleASP1_vbs.asp állomány az InterDev-ben

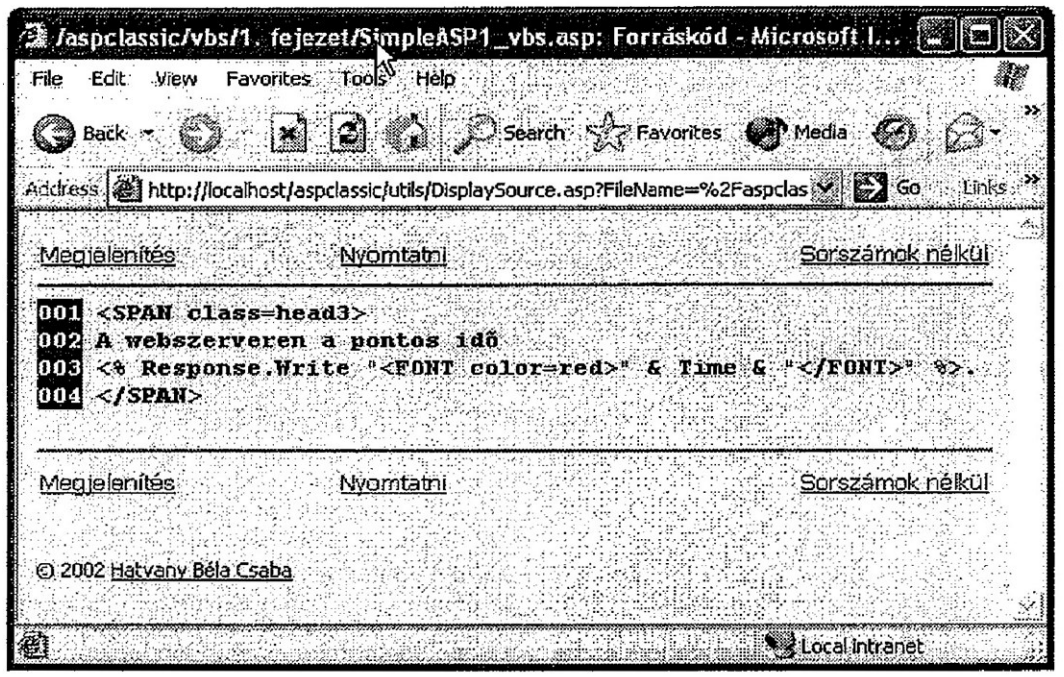
Nézzük meg a `SimpleASP1_vbs.asp` állományt a szövegszerkesztőben. A megjelenített ASP lap középső részének megfelelő kódrészletét két sor között találjuk meg:

```
<!-------------------------------------------------------->
<SPAN class=head3>
A webszerveren a pontos idő
<% Response.Write "<FONT color=red>" & Time & "</FONT>" %>.
</SPAN>
<!-------------------------------------------------------->
```

Az idézett kódrészlet első és utolsó sora egy HTML megjegyzésnek felel meg, nincs semmi kihatása a lap végrehajtására és egyedüli célja csupán annyi, hogy a forráskódot három részre osztja fel. Ez a három rész a megjelenített ASP lap három részének felel meg.

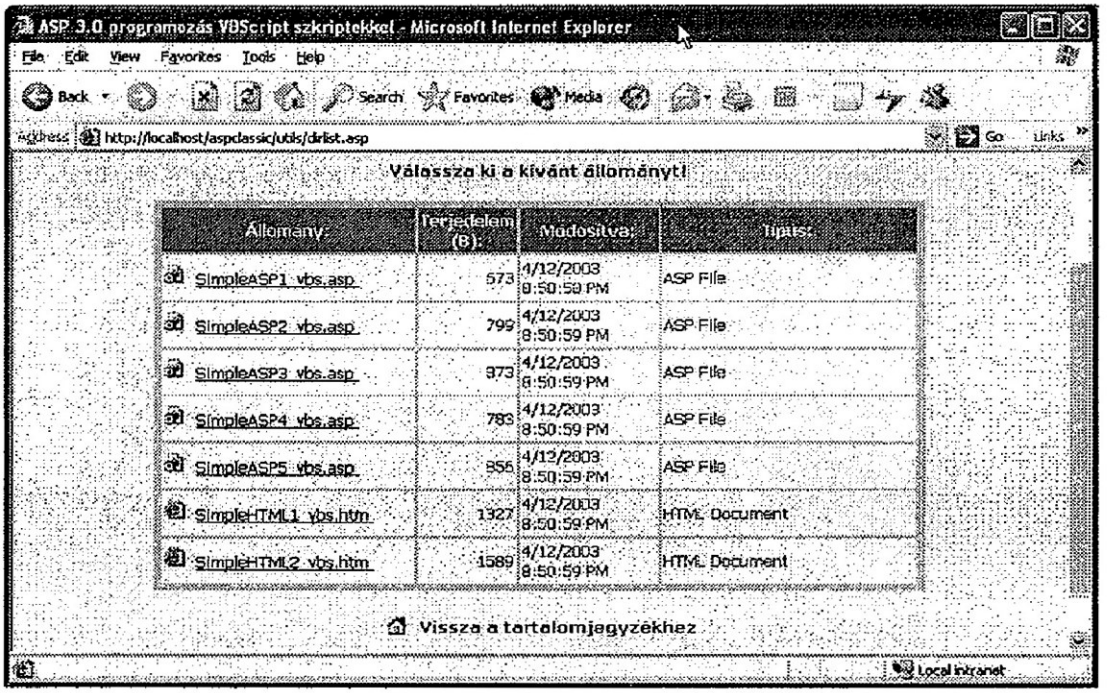
Most nézzük meg a böngészőben megjelenített ASP lap harmadik részét, a második vízszintes vonal alatti kapcsolatot, melynek szövege Forráskód. Ha erre a kapcsolatra kattintunk, akkor a böngészőben megjelenik az előbbi ASP lap forráskódja (1.23. ábra).

Ez a lap is három részből áll. A második rész az előző lap második részének megfelelő forráskódot mutatja. Az első és a harmadik rész a Megjelenítés kapcsolatból áll, mellyel visszahívhatjuk az eredeti ASP lapot.



1.23. ábra A SimpleASP1_vbs.asp ASP lap forráskódja a böngészőben

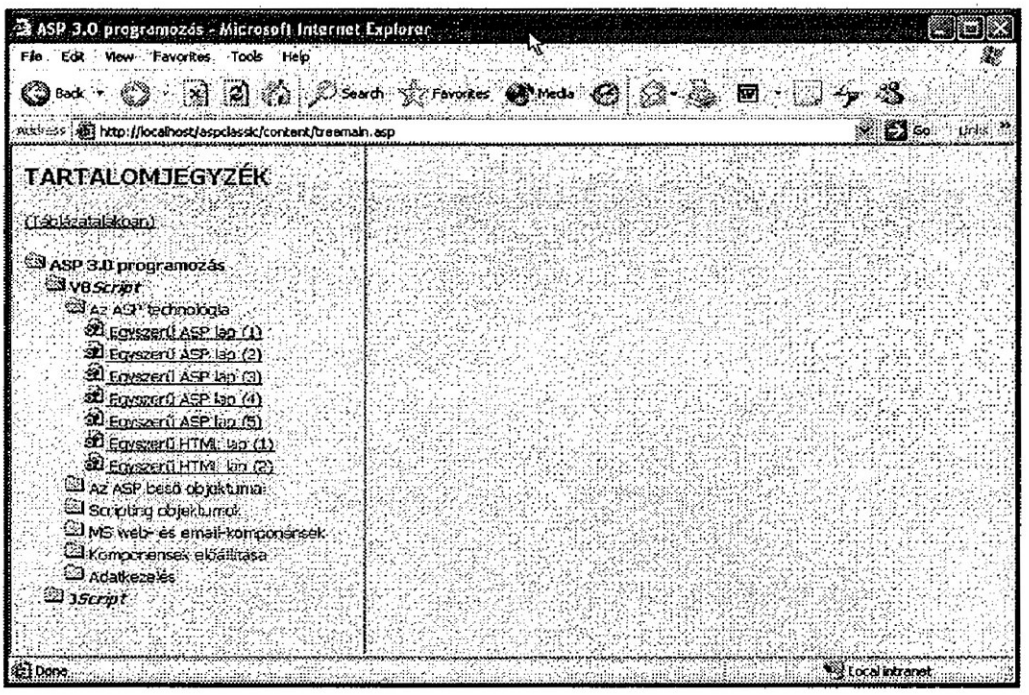
Van egy harmadik lehetőségünk is a példalapokat a szerverről lehívni.



1.24. ábra Az első fejezethez tartozó példák listája

Indítsuk ismét el az *ASPClassic* alkalmazást, mindegy hogy az MMC konzolból vagy az **Internet Explorer**-rel. Az alkalmazás kezdőlapját az 1.15. ábrán láttuk. Ezen a lapon kiválaszthatjuk a szkriptnyelvet és a fejezetet vagy a szakaszt, amelynek a példáit meg akarjuk tekinteni. Ha ezután megnyomjuk a lapon többször elhelyezett **Megnézni** parancsgombot, akkor az alkalmazás megmutatja a kiválasztott részhez tartozó példák listáját. Az 1.24. ábra az első fejezethez tartozó VBScript példákat mutatja meg. Ha

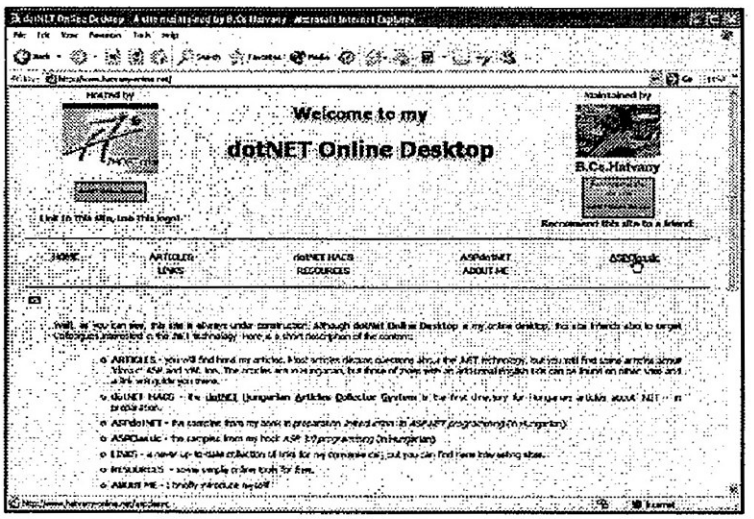
ezen a lapon a `SimpleASP1_vbs.asp` állományra kattintunk, akkor az 1.20. ábrán látott lap jelenik meg.



1.25. ábra Az első fejezethez tartozó példák listája menüalakban

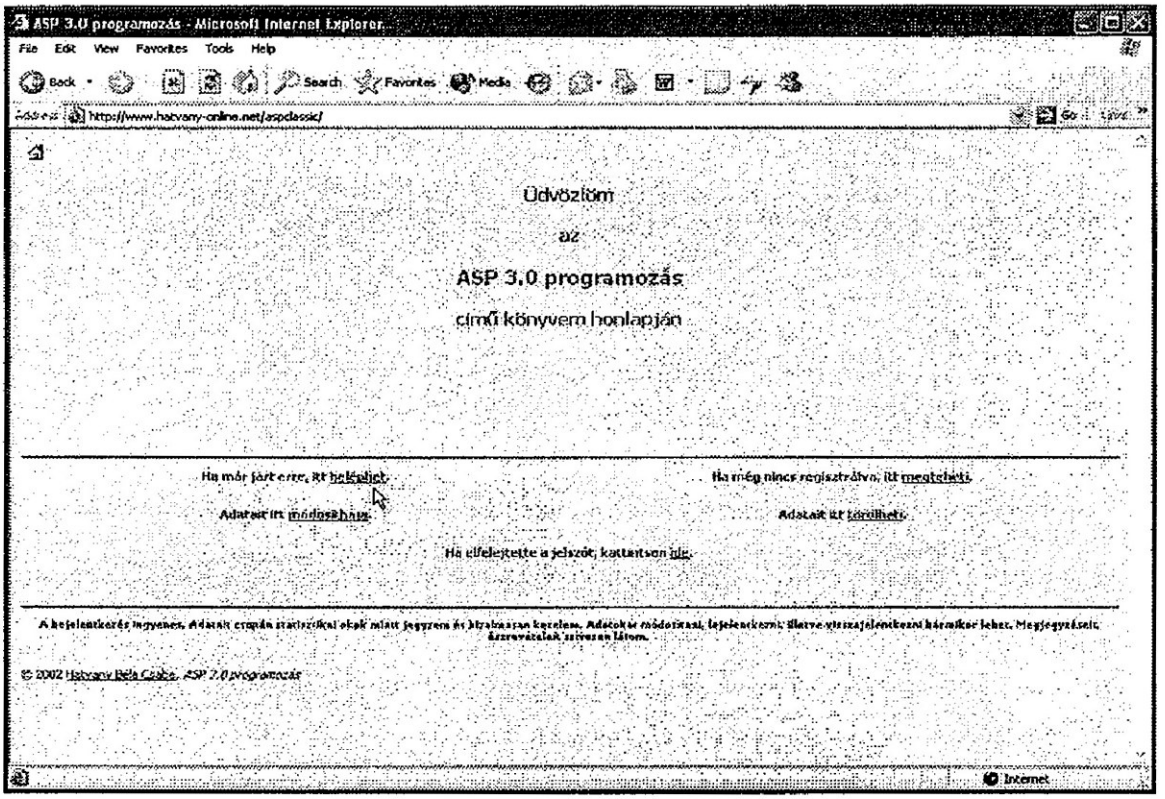
Szintén az alkalmazás 1.15. ábrán látható kezdőlapjáról kiindulva a Menüalakban hivatkozásra kattintva menüalakban megjeleníthetjük a példák listáját (1.25. ábra). A listából kiválaszthatjuk a kívánt példát.

A kedves Olvasó „élőben" is megtekintheti a szemléltető ASP lapokat. Ehhez szükséges egy Internet kapcsolat. A `http://www.hatvany-online.net` cím alatt található a webhelyem, melynek kezdőlapján az **ASPClassic** menü elvezet a példalapokat tartalmazó webalkalmazáshoz (1.26. ábra).



1.26. ábra A szerző weblapja az ASPClassic menüponttal

Mielőtt a látogató eljutna a szemléltető példákhoz, be kell jelentkeznie az email címével (1.27. ábra). Az *ASPClassic* webalkalmazás a megadott címre elküld egy jelszót, amit később meg lehet változtatni és amivel a látogató beléphet a webhely azon részére, ahol a példák találhatók. Ettől a ponttól kezdve a webalkalmazás ugyanúgy működik és (kisebb eltérésektől eltekintve) ugyanazokat a szemléltető példákat tartalmazza, mint a letöltött ZIP állományból a lokális szerverre telepített változat.



1.27. ábra Az ASPClassic webalkalmazás bejelentkező lapja

A többi ASP példa, függetlenül attól, hogy VBScript vagy JScript példa, ugyanígy van felépítve. Majdnem mindegyik VBScript példának megvan a JScript megfelelője a JS mappán belül, mely ugyanúgy van felépítve, mint a VBS mappa. Az így szerkesztett példák könnyen kezelhetők és ha később valamit keresünk, hamar megtaláljuk. Mindehhez még hozzájárul a 2. fejezettől kezdve használt ASP állományok elnevezése is, amiről később még szó lesz. Itt csak annyit jegyezzünk meg, hogy a VBScript és a JScript példaállományok elnevezése csak a `_vbs`, illetve `_js` utótagban különböznek. A `SimpleASP1_vbs.asp` állománynak, amelyik az `ASPClassic\VBS\1. fejezet` mappában található, megfelel a `SimpleASP1_js.asp` állomány, amelyik az `ASPClassic\JS\1. fejezet` mappában található.

A továbbiakban az ASP technológiára vonatkozókat csak a VBScript példák segítségével fogom tárgyalni. Mindent, amit látni fogunk, ugyanúgy elérhető a JScript szkriptnyelv használatával is. Amint nemsokára látni fogjuk, az ASP tökéletesen „érti" mindkét szkriptnyelvet és mindkét szkriptnyelv képes az ASP bármelyik alkotóelemét, illetve ezek tulajdonságait és metódusait elérni.

1.3 Miként működik?

Mostanáig többször használtuk az általánosan ismert szerver és a kliens fogalmakat. Mielőtt megvizsgálnánk az ASP technológia működését, pontosítsuk ezeket a fogalmakat!

Szervernek vagy kiszolgálónak, néha webszervernek vagy webkiszolgálónak nevezzük egyaránt a számítógépet, amin a webszolgáltatásokat nyújtó szoftver fut vagy magát a webszolgáltatásokat nyújtó szoftvert. Esetünkben a szerver az **Internet Information Services 5.0** (IIS), illetve a Windows 2000/XP Professional vagy Windows 2000 Server rendszerrel ellátott számítógép, amin az IIS fut.

Kliensnek vagy ügyfélnek, néha webkliensnek vagy webügyfélnek nevezzük egyaránt a böngésző programot, ami a webszolgáltatásokat igénybe veszi vagy magát a számítógépet, amin a böngésző fut. Esetünkben a kliens az **Internet Explorer 5.0** vagy újabb változata, illetve a Windows 2000/XP Professional rendszerrel ellátott számítógép, amin a böngésző fut.

Ha ugyanazon a gépen fut az IIS és a böngésző, akkor ugyanaz a gép tölti be a szerver és a kliens szerepét. Tanulmányi célokra egy ilyen felállítás minden igénynek megfelel és ezt a felépítést fogom a továbbiakban feltételezni.

1.3.1 Egy ASP lap felépítése és végrehajtása

Térjünk most vissza a legutoljára megjelenített ASP lapra és vegyük szemügyre a lap forráskódját! Nyissuk ki a `SimpleASP1_vbs.asp` állományt a programszerkesztőnkkel (1.21. ábra vagy 1.22. ábra)!

Mindjárt az elején szembetűnik, hogy az állomány két kód „keveréke": a HTML kódba VBScript „kódfoszlányok" vannak beültetve. Tényleg, egy ASP állomány tartalmazhat HTML kódot és ezen belül majdnem bárhol, a `<%` és `%>` elválasztó kódjelektől körülvéve, VBScript vagy JScript kódrészletek szerepelhetnek. Természetes, egy állományon belül a `<%` és `%>` elválasztók között csak az egyik vagy a másik szkriptnyelvet használhatjuk, nem mindkettőt egyszerre.

Az ASP alapértelmezett szkriptnyelve a VBScript. Ha máskép nem rendelkezünk, ASP a `<%` és `%>` között levő kódot VBScript kódnak tekinti. A használt szkriptnyelvet az úgynevezett ASP direktívával (*directive*) határozhatjuk meg.

Egy ASP lap tartalmazhat egy ASP végrehajtási direktívát (*processing directive*). Az ASP végrehajtási direktívának egy `.asp` állományban mindig az első sorban a `<%@`, illetve `%>` elválasztójelek között kell kezdődnie. A végrehajtási direktíva több sajátos

(rész)direktívából állhat, melyek az ASP lap programozási nyelvére, a számok és dátum formázására és más elemekre vonatkozó végrehajtási módját határozzák meg.

A végrehajtási direktíva szintaxisa a következő:

```
<%@
CODEPAGE="codepage"
ENABLESESSIONSTATE="True|False"
LANGUAGE="language"
LCID="lcid"
TRANSACTION="transaction"
%>
```

A CODEPAGE direktíva meghatározza a lapon használt karakterkészletet. Értéke egy egészszámot megadó karakterlánc, amely egyértelműen azonosítja a karakterkészletet. A közép-európai ISO karakterkészlet azonosítója 28592, a Windows karakterkészlet azonosítója 1250. A letöltött ZIP állományban a Függelékek\html mappában található Charsets in Microsoft Internet Explorer 5.htm állomány tartalmazza a használt karakterkészletek azonosítóját.

Az ENABLESESSIONSTATE direktíva meghatározza a munkamenet-állapot használását. Lehetséges értékei **True** és **False**. Az alapértelmezett érték **True**, mely a munkamenet-állapot használatát biztosítja. **False** érték esetében a munkamenet-állapot ki van kapcsolva. A munkamenettel a 2. fejezetben részletesen foglakozunk.

Az ASP lap programozási nyelvét a LANGUAGE direktíva rögzíti. Alapértelmezett értéke VBScript. A LANGUAGE direktívával megadott szkriptnyelv minden további <% és %> között szereplő kódra vonatkozik. A LANGUAGE direktívával rögzített programozási nyelv mellett más nyelvben írt szkripteket is használhatunk. Ezeket a kódrészleteket a használt szkriptnyelv megadásával a <SCRIPT LANGUAGE = "language" > és </SCRIPT> elválasztók közé kell írnunk.

Az LCID direktíva a helyazonosítót (*locale identifier*) adja meg. Értéke egy egészszám, mely a helyi (természetes) nyelvet azonosítja. A magyar nyelv azonosítója a 1038 decimális szám. A használt helyazonosítókat a letöltött állományban található Függelékek\html\Locale ID.htm állomány tartalmazza.

A TRANSACTION direktíva megadja, milyen tranzakció környezetben (*transaction context*) történjen az ASP lap végrehajtása. Értékei Disabled, Requires_New, Required, Supported és Not_Supported. Alapértelmezett értéke Disabled. A tranzakciós ASP lapokkal (*transactional ASP page*) ebben a könyvben nem foglalkozunk. Ennek a direktívának mindig az alapértelmezett értékét fogjuk használni, ami tranzakció nélküli ASP végrehajtást eredményez.

A példáinkban a következő végrehajtási direktívát fogjuk használni:

```
<%@
LANGUAGE="VBScript" CODEPAGE="28592" LCID="1038"
%>
```

illetve

```
<%@
LANGUAGE="JScript" CODEPAGE="28592" LCID="1038"
%>
```

Ezt a végrehajtási direktívát az első példánk forráskódjában az állomány első sorában láthatjuk (1.21. ábra és 1.22. ábra). Az állomány többi része HTML kódból és a <% és %> elválasztók által közrezárt VBScript, illetve JScript kódból áll. A végrehajtási direktíva csak az ASP (VBScript vagy JScript) kódra vonatkozik és nem érinti a „tiszta" HTML kódot.

Korlátozzuk figyelmünket a 1.23. ábrán látható kódrészletre. Itt HTML kódba beékelve egy ASP kódrészletet látunk:

```
<% Response.Write Time %>
```

A következő fejezetben részletesen megtárgyaljuk az ASP objektumokat, metódusokat és tulajdonságokat. Most csak annyit jegyezzünk meg, hogy `Response` egy ASP objektum, melynek `Write` egy metódusa. A `Response` objektum `Write` metódusával a `Time` VBScript függvény által visszaadott értéket kiírhatjuk a böngészőben megjelenített lapra. A `Time` függvény a webszerveren érvényes időpontot adja vissza.

1.28. ábra A megjelenített ASP lap forráskódja

Már most nyilvánvaló, hogy a szerverünk képes a HTML és ASP kód között különbséget tenni és ezt a kétféle kódot megfelelőképpen kezelni, úgy hogy végeredményben a megjelenített lap pontosan úgy néz ki, mint ahogy azt elképzeltük. Mielőtt azonban részletesebben megtárgyalnánk miként is kezeli a webszerver az ASP lapokat, nézzük meg még egyszer a megjelenített ASP lap forráskódját. Ezúttal azonban használjuk a böngészőnk **View|Source** menüparancsát. A böngésző meghívja a **Notepad**-et és megmutatja a *megjelenített* ASP lap forráskódját, amint azt az 1.28. ábrán láthatjuk.

Hasonlítsuk össze az 1.28. ábrán látható forráskódot az 1.21. ábrán vagy az 1.22. ábrán látható forráskóddal! A böngésző által megmutatott forráskód „vegytiszta" HTML kód, eltűnt belőle minden „idegen" elem. Ha a kiemelt kódrészletet megvizsgáljuk, akkor látjuk, hogy a `Response.Write` ASP metódus és a `Time` VBScript függvény eltűnt és helyettük az időpont áll:

```
<SPAN class=head3>
A webszerveren a pontos idő
<FONT color=red>8:35:28 PM</FONT>.
</SPAN>
```

(Az idő most természetesem más, mint amit a 1.20. ábrán láttunk, mert a lap két felhívása különböző időpontban történt.) Most már könnyebb megérteni miként működik az ASP.

Az ügyfélgép, pontosabban a kliensgépen futó böngésző, egy kéréssel fordul a webszerverhez: egy ASP lap, esetünkben a

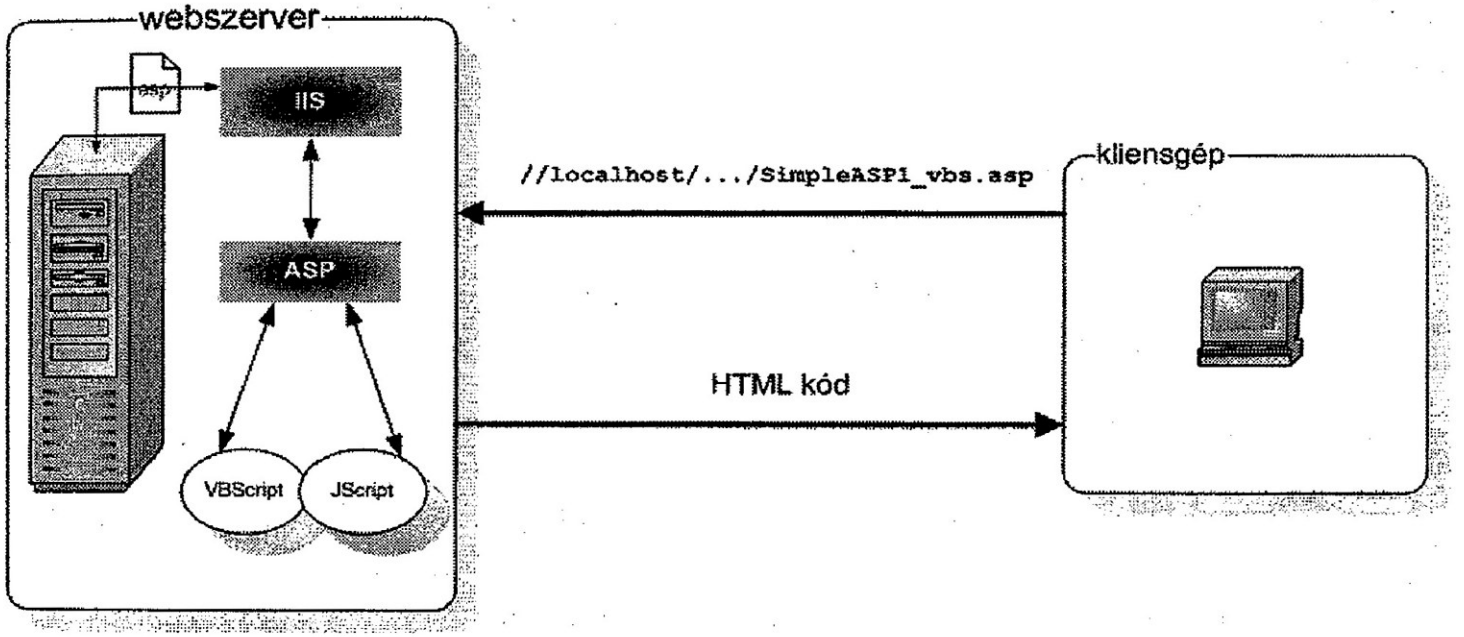
```
//localhost/ASPClassic/vbs/1. fejezet/SimpleASP1_vbs.asp
```

lap megjelenítését kéri. A kliens kérését a webszerver webszolgáltató programja, az IIS fogadja. Az IIS megkeresi a szerveren a kért lapot. Ha a kért lapnak a kiterjesztése `.asp` és ha a lapot az IIS a szerveren megtalálja, akkor átadja az ASP *script host*-nak. A script host az `asp.dll` dinamikusan kapcsolt könyvtár, amelyik általában a rendszermappában a `\system32\inersrv` mappában található. Ez a DLL felelős az ASP lap helyes értelmezéséért. A szkriptgazdának eljuttatott állományt a script host elolvassa, azonosítja a használt szkriptnyelvet és a szkripteket, és a megfelelő szkriptértelmezőhöz küldi: a VBScript szkripteket a VBScript értelmezőgéphez, a JScript szkripteket a JScript értelmezőgéphez. Ezek a szkript-értelmezőgépek a szerveren telepített szoftverek. Elméletileg a VBScript és JScript értelmezőkön kívül más szkriptnyelveket értelmező gépek is telepíthetők a szerverre, ez a kettő azonban mindig jelen van. A szkriptértelmező átalakítja a kapott kódot HTML kódra és visszajuttatja a szkriptgazdának. Most a script host feladata a szkriptértelmezés eredményként kapott HTML kódot az ASP lapban levő HTML kódba, a kellő helyre „beleszúrni". Az eredmény kizárólag HTML kódból áll (Itt eltekintek attól, hogy az eredeti ASP lap esetleg kliensoldali szkriptet is tartalmazhat, amit a szerver érintetlenül hagy. Ezzel az esettel a következő szakaszban foglalkozunk.). Ezt a HTML kódot a script host most

visszaadja az IIS-nek, mely gondoskodik arról, hogy a HTML kódáramlat (*code stream*) válaszként eljusson a kérést intéző ügyfélhez, azaz a böngészőhöz. Végül a böngésző feladata, hogy a válaszként kapott "tiszta" HTML kódáramlatot megjelenítse.

A leírt folyamatot az 1.29. ábra szemlélteti. Természetesen itt csak vázlatosan írtam le az egész folyamatot, ami a kliensgép lapkérése és a lap megjelenítése között történik. Számos részletet, mint például a gyorstár használata (*cashing*), ami számunkra most nem fontos, kihagytam.

Úgyszintén nem esett szó az ASP lapon esetleg létező szerver- és kliensoldali szkriptekről. Olyan szkriptekről van szó, melyeket az ismert `<SCRIPT LANGUAGE = VBScript RUNAT=SERVER>`, illetve `<SCRIPT LANGUAGE = VBScript >` és `</SCRIPT>` között írhatunk egy ASP lapba. A következő szakaszban megvizsgáljuk miként hajtja végre az ASP az ilyen szkripteket.



1.29. ábra Egy ASP lap kérése a szervertől és megjelenítése a kliensgépen

1.3.2 Szerver- és kliensoldali szkriptek

Az előbbi szakaszban megemlítettem, hogy egy ASP lapon belül szerveroldali és kliensoldali szkripteket is használhatunk. Mielőtt részletesen foglalkoznánk ezekkel a szkriptekkel, pontosítsuk a használt kifejezéseket!

Az ASP lapokon belül használt VBScript vagy JScript kódot, amit `<%` és `%>` elválasztók közé írunk, ASP szkriptnek vagy ASP kódnak fogom nevezni. Ha szükséges a

szkriptnyelv megnevezése, akkor VBScript vagy JScript (szkriptnyelvben írt) ASP szkriptekről vagy ASP kódról lesz szó.

Az ASP szkripteken kívül egy ASP lap VBScript vagy JScript szkripteket is tartalmazhat. Olyan kódrészletről van szó, amit a <SCRIPT ...> és </SCRIPT> elválasztók közé írunk. A <SCRIPT ...> utasítás a használt szkriptnyelvet a LANGUAGE = *szkriptnyelv* részutasítással adja meg, ahol *szkriptnyelv* "VBSCRIPT" vagy "JSCRIPT", illetve más, például "JAVASCRIPT", szkriptnyelvet azonosító karakterlánc lehet. Mi csak a VBScript és JScript szkriptnyelveket használjuk. Egy VBScript vagy JScript szkript vázlata tehát

```
<SCRIPT LANGUAGE = ...>
' ide írjuk a kódot
</SCRIPT>
```

Ha a <SCRIPT ...> utasítás nem tartalmaz más elemet, akkor egy kliensoldali VBScript, illetve JScript szkriptről beszélünk. Ha viszont a <SCRIPT ...> utasítás a RUNAT = SERVER részutasítást is tartalmazza, akkor egy szerveroldali VBScript, illetve JScript szkriptről beszélünk:

```
<SCRIPT LANGUAGE = ... RUNAT = SERVER>
' ide írjuk a kódot
</SCRIPT>
```

Fontos megjegyezni, hogy az ASP lapokon, ellentétben a HTML lapokkal, a RUNAT = CLIENT utasítás egy hibát generál.

Összefoglalva: az ASP lapokon

- VBScript vagy JScript szerver- vagy kliensoldali szkripteket használhatunk;
- a szerveroldali szkripteket RUNAT = SERVER utasítással kell ellátni;
- amennyiben a RUNAT = SERVER utasítás elmarad, úgy egy kliensoldali szkriptről van szó;
- a RUNAT = CLIENT utasítás egy hibát generál.

Nézzük most meg, miként értelmezi az ASP a VBScript és JScript szerver-, illetve kliensoldali szkripteket.

Az \ASPClassic\VBS\1. fejezet\SimpleASP1_vbs.asp állományban helyettesítsük a

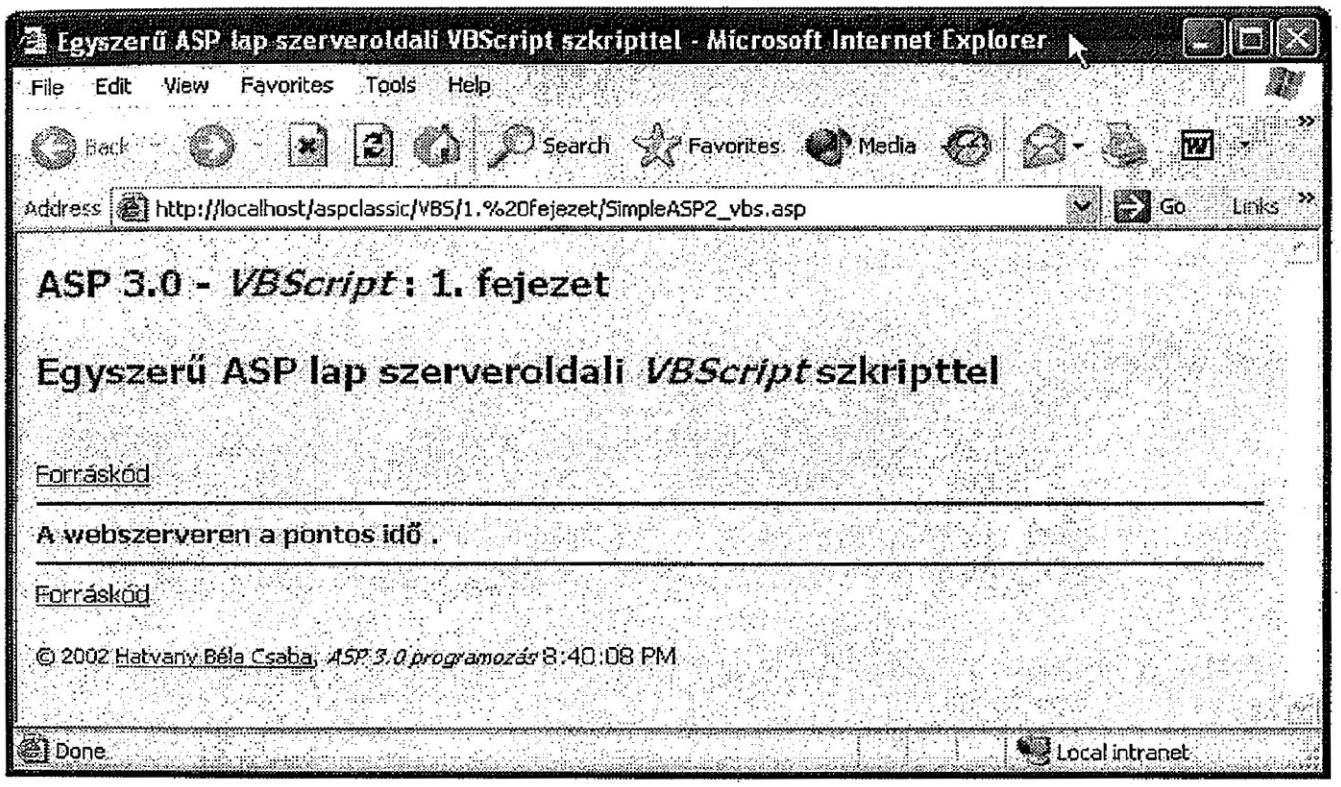
```
<SPAN class=head3>
A webszerveren a pontos idő
<% Response.Write "<FONT color=red>" & Time & "</FONT>" %>.
</SPAN>
```

kódrészletet a következő kódrészlettel

```
<SPAN class=head3>
A webszerveren a pontos idő
<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>
Response.Write "<FONT color=red>" & Time & "</FONT>"
</SCRIPT> .
</SPAN>
```

és mentsük a módosított állományt a `SimpleASP2_vbs.asp` állományba. (Ha nem akarjuk az eredeti állományt módosítani, akkor nézzük meg a letöltött ZIP állományban levő állományt.) Ez a módosítás az eredeti ASP szkriptet egy szerveroldali VBScript szkripttel helyettesíti.

Jelenítsük meg a böngészőnkben az így létrehozott ASP lapot (1.30. ábra)!



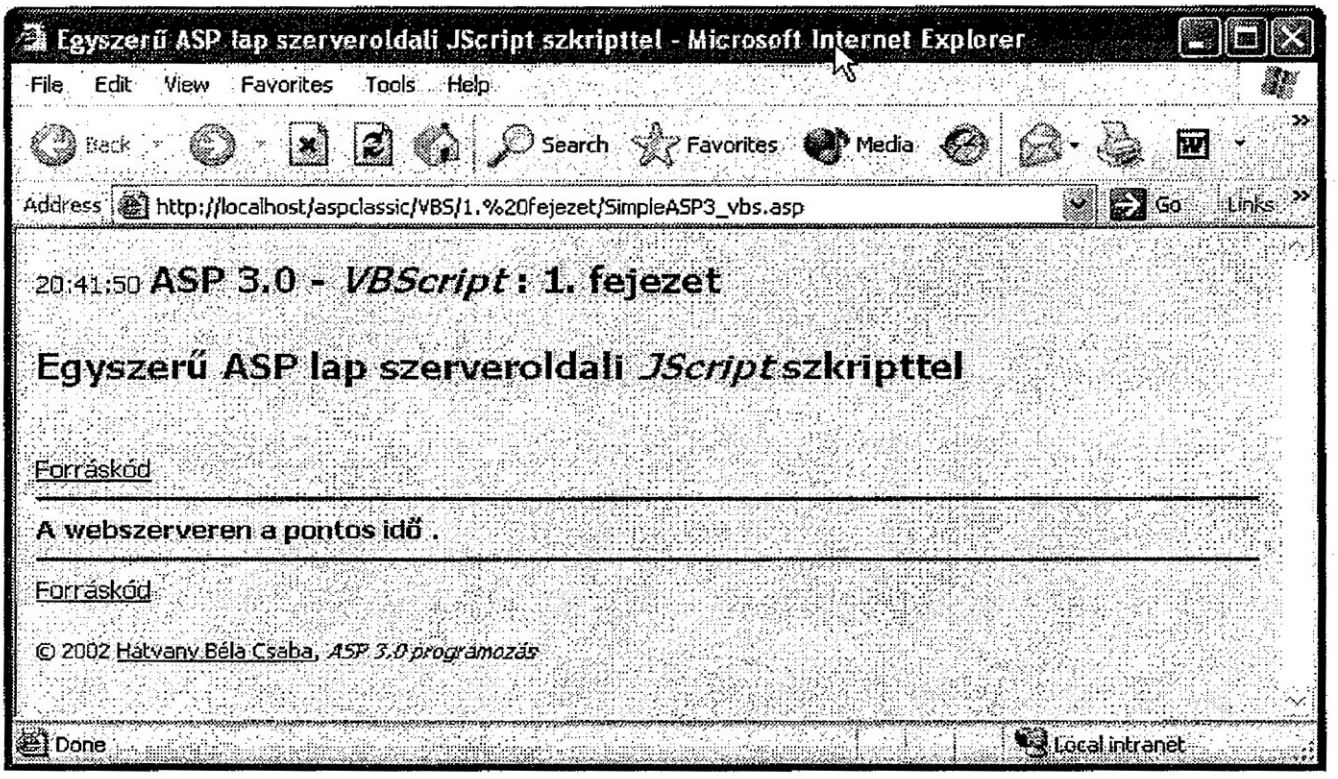
1.30. ábra Egy szerveroldali VBScript szkript értelmezése egy ASP lapban

Az eredmény nem megfelelő: az időpont nem a kívánt helyre, hanem a lap végére került. (A megváltozott időpontnak természetesen nincs különösebb jelentősége.)

Mielőtt bármilyen következtetést levonnánk, nézzük meg, milyen eredményt kapunk, ha egy szerveroldali JScript szkriptet használunk. Helyettesítsük a legutolsó VBScript szkriptet egy megfelelő JScript szkripttel, vagy nézzük meg a `SimpleASP3_vbs.asp` állomány forráskódját:

```
<SPAN class=head3>
A webszerveren a pontos idő
<SCRIPT LANGUAGE=JSCRIPT RUNAT=SERVER>
var d = new Date();
Response.Write('<FONT color=red>' + d.getHours() + ':' +
d.getMinutes() + ':' + d.getSeconds() + '</FONT>');
</SCRIPT>
.
</SPAN>
```

A böngészőben megjelenített állományt az 1.31. ábrán láthatjuk.



1.31. ábra Egy szerveroldali JScript szkript értelmezése egy ASP lapban

Az eredmény most sem megfelelő: az időpont nem a kívánt helyre, hanem a lap elejére került.

Ha ugyanezeket a módosításokat a `SimpleASP1_js.asp` állománnyal is elvégezzük és az így kapott `SimpleASP2_js.asp`, illetve `SimpleASP3_js.asp` állományt megjelenítjük a böngészőben, az eredmény ugyanaz és ekképpen foglalhatjuk össze:

- egy ASP lapon alkalmazott szerveroldali szkript végrehajtási sorrendje meghatározhatatlan;
- a jelen implementációban egy VBScript (illetve JScript) szkriptnyelv segítségével írt ASP lapon egy szerveroldali VBScript (illetve JScript) szkript az egész lap értelmezése után lesz értelmezve, míg egy szerveroldali JScript (illetve VBScript) szkript az egész lap értelmezése előtt lesz értelmezve.

Ha a kérdéses lapok forráskódját a böngészőnk segítségével nézzük meg, azaz megnézzük a webszervertől a böngészőnek visszaküldött kódot, akkor ott az időpontot látjuk: egyszer a kód végén, egyszer pedig a kód elején. Ez bizonyítja, hogy a szerveroldali szkripteket tényleg a szerver értelmezte. A valóságban a szerver kétszer olvassa el a böngésző által kért ASP lapot: először az ASP szkriptrészleteket azonosítja, azaz a `<%` és `%>` elválasztók között levő kódot értelmezi, majd a HTML kódba a megfelelő helyre beilleszti a kiértékelt ASP kódot. Csak ezután, a lap második olvasásakor azonosítja a szerveroldali szkripteket. Ezeket a megfelelő értelmezőhöz küldi és a visszakapott eredményt, attól függően, hogy a szerveroldali szkriptek ugyanazon a nyelven vannak írva, mint az ASP kód vagy nem, a HTML kód végére vagy elejére írja.

Természetesen, megállapításunk csak a VBScript és JScript nyelvekre és a jelenlegi implementációra érvényes.

Ez egy bizonytalansági tényezőt jelent az ASP lapok programozásában. Ezért ASP lapokon, ha nem feltétlenül szükséges, nem fogunk szerveroldali szkripteket használni.

Most nézzük meg, mi a helyzet a *kliensoldali* szkriptekkel.

Induljunk el a már ismert `SimpleASP1_vbs.asp` állományból és helyettesítsük az előbbi kódrészletet ezúttal egy kliensoldali szkripttel, egyszer VBScript szkriptnyelvet használva:

```
<SPAN class=head3>
Az ügyfélgépen a pontos idő
<SCRIPT LANGUAGE=VBSCRIPT >
document.write "<FONT color=red>" & Time & "</FONT>"
</SCRIPT>.
</SPAN>
```

majd pedig JScript szkriptnyelvet használva:

```
<SPAN class=head3>
Az ügyfélgépen a pontos idő
<SCRIPT LANGUAGE=JSCRIPT >
var d = new Date();
document.write('<FONT color=red>' + d.getHours() + ':' +
d.getMinutes() + ':' + d.getSeconds() + '</FONT>');
</SCRIPT>.
</SPAN>
```

Figyeljük meg, hogy mindkét esetben nem szerveroldali `Response` objektumot, hanem a kliensoldali `document` objektumot használtam. Az így kapott eredményt a `SimpleASP4_vbs.asp` és `SimpleASP5_vbs.asp` állományban látjuk.

Az eredmény mindkét esetben helyes, de tudnunk kell, hogy ebben az esetben a kliensgépen érvényes időpont lett megjelenítve. Ha megnézzük a böngésző segítségével a megjelenített állományok forráskódját, akkor láthatjuk, hogy a szerver érintetlenül hagyva elküldte a kliensoldali szkripteket a böngészőnek és végrehajtásukat a böngészőnek engedte át.

Hasonló módosításokat végezhetünk a `SimpleASP1_js.asp` állománnyal is, hogy a megfelelő `SimpleASP4_js.asp`, illetve `SimpleASP5_js.asp` állományhoz jussunk és ezek megjelenítése a böngészőben hasonló eredménnyel jár.

A tapasztaltakat így foglalhatjuk össze:

- az ASP lapokon alkalmazott kliensoldali szkripteket a webszerver érintetlenül hagyja és elküldi kliensnek, hogy értelmezze;
- a kliensoldali szkriptek sikeres értelmezése a kliens „nyelvtudásától" függ.

Mielőtt befejeznénk ezt a bevezető fejezetet, nézzük meg, hogy kezelik a HTML lapok a kliens-, illetve szerveroldali szkripteket. Ha ez nem is tartozik szorosan a tárgyalt témához, érdemes tudni az eredményt.

Legelőször emlékezzük arra, hogy az ASP csak a `.asp` kiterjesztésű lapokat kezeli. Ez azt jelenti, hogy a HTML lapokat a szerver érintetlenül hagyva elküldi a kérő ügyfélnek.

A `SimpleHTML1_vbs.htm`, illetve `SimpleHTML1_js.htm` állományok megmutatják, hogy a HTML lapon alkalmazott kliensoldali szkriptet a böngésző értelmezi és, amennyiben a szkript nyelvét „érti", minden nehézség nélkül kellőképpen végrehajtja. Egy HTML lapon egy kliensoldali szkript váza a következő:

```
<SCRIPT LANGUAGE = ... RUNAT = CLIENT>
' ide írjuk a kódot
</SCRIPT>
```

ahol a `RUNAT = CLIENT` részutasítást elhagyhatjuk, mivelhogy ez az alapértelmezésnek felel meg (s mint máris látni fogjuk, más értelmezés nem is létezik).

Egy HTML lap nem „ismeri" a szerveroldali szkripteket, ami természetes is, mivelhogy a szerver egyszerűen csak elküldi minden értelmezés nélkül a kért lapot a kliensnek. Ugyanakkor tudjuk, hogy a böngésző HTML értelmezője sokkal elnézőbb, mint az ASP: sok szintaxis hibát elnéz nekünk és egyszerűen figyelmen kívül hagyja.

Ezzel magyarázható, hogy az alábbi két kódrészlet, noha alapjában véve szintaktikailag hibás, mégis bizonyos esetekben végrehajtható.

Ezt az esetet a SimpleHTML2_vbs.htm

```
<H2>
A pontos idő
<!--
A HTML lapok nem ismerik a szerveroldali szkripteket,
ezért nem veszik figyelembe a RUNAT=SERVER utasítást.
EZ AZ OLDAL EGY HIBÁT GENERÁL: nem ismeri fel a szerveroldali
Response objektumot. Ha a Response objektumot a kliensoldali
document objektummal helyettesítjük, akkor a szkript kliensoldali
szkriptként lesz értelmezve
-->
<SCRIPT LANGUAGE=VBSCRIPT RUNAT=SERVER>
Reponse.Write Time
'document.write Time
</SCRIPT>
.
</H2>
```

illetve a SimpleHTML2_js.htm

```
<H2>
A pontos idő
<!--
A HTML lapok nem ismerik a szerveroldali szkripteket,
ezért nem veszik figyelembe a RUNAT=SERVER utasítást.
EZ AZ OLDAL EGY HIBÁT GENERÁL: nem ismeri fel a szerveroldali
Response objektumot. Ha a Response objektumot a kliensoldali
document objektummal helyettesítjük, akkor a szkript kliensoldali
szkriptként lesz értelmezve
-->
<SCRIPT LANGUAGE=JSCRIPT >
var d = new Date();
Response.Write(d.getHours()+":"+d.getMinutes()+":"+d.getSeconds()
);
//document.write(d.getHours()+":"+d.getMinutes()+":"+d.getSeconds
());
</SCRIPT>
.
</H2>
```

állomány mutatja be.

A kódba fűzött kommentár elmagyarázza a szkript értelmezését.

2. Az ASP belső objektumai

Mint láttuk, az ASP objektumokat három osztályba sorolhatjuk: az ASP belső objektumai, az ASP VBScript és JScript objektumai vagy szkriptobjektumok és a külső, de az ASP-be beépíthető objektumok.

Ebben a fejezetben az ASP belső (*intrinsic*) objektumait fogjuk megismerni. Ezeket az objektumokat nem kell létrehoznunk. Létrehozásukról maga az ASP gondoskodik. Az ASP összesen hat belső objektummal rendelkezik:

- `Server`
- `Application`
- `Session`
- `Request`
- `Response`
- `ASPError`

Az ASP belső objektumai képezik az ASP vázát, amire az objektumok metódusai, tulajdonságai, gyűjteményei és eseményei felépülnek, hogy együtt alkossák az ASP szolid testét. A belső objektumok mindegyike specifikus feladatokat lát el és ehhez mérten megfelelő metódusokkal, tulajdonságokkal, gyűjteményekkel és eseményekkel rendelkeznek.

A `Server` objektum a webkiszolgálót vagy webszervert képviseli. Az `Application` objektum a webalkalmazásnak felel meg, mely, mint látni fogjuk, a webalkalmazás alapmappájában és ennek almappáiban található ASP kód összességéből áll. A `Session` objektum a webszerver és a kliens vagy ügyfél között fennálló kapcsolatot, az úgynevezett munkamenetet képviseli. A `Request` és `Response` objektumok a webszerver és kliens között folytatott „párbeszéd" kérés-, illetve válaszüzenetének felelnek meg. Végül az `ASPError` objektum a hibakezelést segíti elő.

Ebben a fejezetben ezeket az objektumokat fogjuk megtárgyalni.

2.1 A `global.asa` állomány

Az `Application` és a `Session` objektumokat később részletesen megtárgyaljuk. Jelen pillanatban jegyezzük meg, hogy e két objektumnak két-két eseménye van, melynek a megfelelő eseménykezelő eljárását a webalkalmazás csak egy bizonyos helyén hívhatjuk meg. Ez a szerver által ismert hely, a `global.asa` állomány, mely-

nek az alkalmazás alapmappájában kell lennie. Csak itt ismeri fel a szerver a `global.asa` állományt. A `global.asa` állomány általános felépítése a következő:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server >

' a webalkalmazás indulásakor
Sub Application_OnStart()
'
End Sub

' a munkamenet indulásakor
Sub Session_OnStart()
'
End Sub

' a munkamenet befejezésekor
Sub Session_OnEnd
'
End Sub

' a webalkalmazás befejezésekor
Sub Application_OnEnd
'
End Sub

</SCRIPT>
```

Az állományon belül nem szerepelhet ASP kód `<%` és `%>` között. Ezzel ellentétben egy teljes szkriptet kell tartalmaznia. E szkripten belül definiálhatjuk az `Application` objektum `Application_OnStart` és `Application_OnEnd`, illetve a `Session` objektum `Session_OnStart` és `Session_OnEnd` eseménykezelő eljárásokat, amiről később még szó lesz.

Ha egy webalkalmazáson belül egy másik webalkalmazást definiálunk, akkor mindegyik alkalmazás számára egy saját `global.asa` állományt adhatunk meg. Tudnunk kell azonban, hogy a „belső" alkalmazás állományaira csak a saját `global.asa` állományra érvényes, míg a „külső" `global.asa` állomány csak azokra az állományokra van hatással, melyek a „külső" alkalmazáshoz tartoznak, de ugyanakkor nem tartoznak a „belső" alkalmazáshoz.

A fejezet folyamán találkozni fogunk még a `global.asa` állománnyal és akkor részletesen megismerjük a szerepét.

2.2 A `Server` objektum

A `Server` objektum a szerver környezetével és aktivitásával kapcsolatos feladatokat képes elvégezni a szerveren. A legfontosabb ilyen feladat, más objektumok létrehozása a szerveren. A `Server` objektum nem rendelkezik saját eseménnyel. Hét metódusa van:

- `CreateObject`
- `Execute`
- `GetLastError`
- `HTMLEncode`
- `MapPath`
- `Transfer`
- `URLEncode`

és egy tulajdonsága: `ScriptTimeout`.

2.2.1 A `Server` objektum metódusai

A `Server` objektum talán legfontosabb metódusa a `CreateObject` metódus. Segítségével komponensek, azaz szkriptek vagy alkalmazások által definiált objektumokat tudunk a szerveren létrehozni. Ezeket az úgynevezett külső (*non-intrinsic*) objektumokat létrehozásuk után felhasználhatjuk az alkalmazásban. Ebben rejlik az ASP sokrétűsége, flexibilitása és nagy hatékonysága. A `CreateObject` metódus szintaxisa a következő:

```
Set obj = Server.CreateObject(id)
```

ahol *id* egy karakterlánc (*string*) típusú objektumazonosító és `obj` egy objektumreferencia vagy objektumhivatkozás. Egy objektumreferencia egy ASP (VBScript vagy JScript) változó, mely egyértelműen egy létrehozott objektumot referál, más szóval, az objektumra hivatkozik. Az *id* objektumazonosító lehet egy osztályazonosító (`ClassID`), mint például

```
"{clsid:AB675....A1}"
```

vagy egy programazonosító (`ProgID`), mint például

```
"MSWC.Counter"
```

Az osztály- és programazonosítóról később többet fogunk megtudni, most csak annyit jegyezzünk meg, hogy segítségükkel egy objektumot egyértelműen azonosíthatunk.

A `CreateObject` metódust meghívhatjuk egy ASP lapban vagy az alkalmazás `global.asa` lapjában. Ha egy objektumot egy (közönséges, nem a `global.asa`) ASP lapban hozunk létre, akkor hatóköre (*scope*) nem terjed a lapon túl. Ilyen objektumot oldalszintű vagy lapszintű objektumnak (*page-level object*) is nevezzük. Egy lapszintű objektum megszűnik létezni, miután a lap megszűnt és a lap ismételt lehívásakor az objektum egy új példánya jön létre.

Ha az objektumot az alkalmazás `global.asa` lapjában hozzuk létre, akkor két lehetőségünk van. Létrehozhatunk egy alkalmazás-hatókörű (*application-scope*) vagy egy munkamenet-hatókörű (*session-scope*) objektumot. Az webalkalmazásra és a munkamenetre később még visszatérünk.

Webalkalmazás-hatókörű objektum vagy alkalmazásszintű objektum (*application-level object*) létrehozására a `Server` objektum `CreateObject` metódusát a `global.asa` lapban az `Application_OnStart` eseménykezelő eljárásból kell meghívnunk. Egy ilyen objektum az alkalmazás indulásával jön létre, az alkalmazás bármelyik lapjáról elérhető, és ha előbb nem szüntetjük meg, az alkalmazás végéig létezik. Az alkalmazás megszűntével az alkalmazásobjektumok megszűnnek létezni. Ennek ellenére tanácsos őket az `Application_OnEnd` eljárásban törölni. Egy alkalmazásobjektumnak csak egy példánya létezhet egy adott alkalmazáson belül és ez minden kliens által elérhető.

Munkamenet-hatókörű objektum vagy munkamenetszintű objektum (*session-level object*) létrehozására a `Server` objektum `CreateObject` metódusát a `global.asa` lapban a `Session_OnStart` eseménykezelő eljárásból kell meghívnunk. Egy ilyen objektum a munkamenet indulásával jön létre, az alkalmazás bármelyik lapjáról elérhető és, ha előbb nem szüntetjük meg, a munkamenet végéig létezik. A munkamenet megszűntével a munkamenetszintű objektumok megszűnnek létezni. Ennek ellenére, úgy mint az alkalmazásszintű objektumok esetében, tanácsos őket a `Session_OnEnd` eljárásban törölni. A munkamenetszintű objektumokat is eléri az alkalmazás folyamán mindegyik kliens, de az alkalmazásszintű objektumoktól eltérően, minden kliens a munkamenetszintű objektum egy külön példányát használja. Más szóval, egy alkalmazáson belül egy adott időpontban legfeljebb annyi munkamenetszintű objektum lehetséges, ahány kliens használja az alkalmazást az adott időpontban.

A `Server` objektum `CreateObject` metódusán kívül, objektumokat a HTML `<OBJECT>` elemmel is létrehozhatunk:

```
< OBJECT ID = objid RUNAT = where SCOPE = level PROGID = id >
< /OBJECT >
```

A különböző attribútumok értékei karakterláncok a következő jelentéssel:

- *objid* az objektumreferencia neve,
- *where* lehet "Server" vagy "Client" és meghatározza, hol hozzuk létre az objektumot,
- *level* lehet "Application", "Session" vagy "Page" és meghatározza az objektum hatókörét,
- *id* az objektum azonosítója.

A különbség a Server objektum CreateObject metódusával és a HTML <OBJECT> elem által létrehozott objektumok között abban áll, hogy a CreateObject metódus azonnal létrehozza az objektumot, míg a HTML <OBJECT> elemmel definiált objektum csak az első hivatkozáskor jön létre.

A Server objektum CreateObject metódusának alkalmazását az Application objektummal foglalkozó szakaszban fogjuk látni.

Vizsgáljuk most meg az Execute metódust! Szintaxisa

```
Server.Execute url
```

ahol *url* egy ASP vagy HTML lap relatív virtuális elérési útja (URL-je). Az Execute metódus átirányítja az alkalmazást az *url* által megadott lapra, amit letölt és beékel a böngészőben megjelenített lapba, majd továbbfolytatja az eredeti lap következő utasításával.

Az /ASPClassic/VBS/2. fejezet/Server/Server.Execute.asp lap tartalmazza az Execute metódust bemutató példánkat Az idevonatkozó kódrészlet a következő:

```
<%
Server.Execute "Server.Execute/testExecute.htm"
Response.Write "<HR>Ezt a sort és az fölötte " & _
"levő vonalat az eredeti <b>.asp</b> állomány írta."
%>
```

Az Execute metóduson kívül a Response objektum Write metódusát is felhasználtuk. Ezt az objektumot részletesen később fogjuk megismerni. Most csak annyit kell tudnunk, hogy a Write metódusával egy szöveget írhatunk ki a böngészőben megjelenített lapra:

```
<% Response.Write "Ezt a szöveget a lapra írjuk" %>
```

A fenti kódsornak van egy rövidített alakja is:

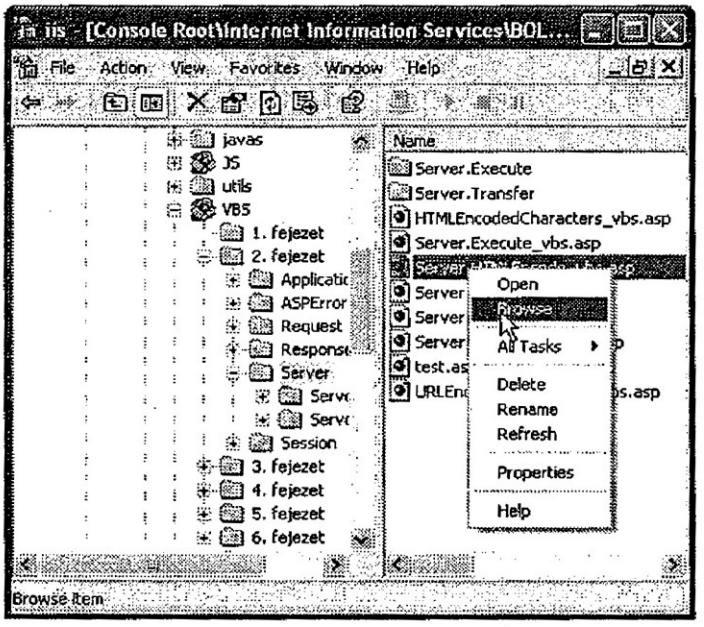
```
<% = "Ezt a szöveget a lapra írjuk" %>
```

amit néha használni fogunk. Ügyelnünk kell, hogy a rövidített alakban a <% és %> jelek között nem szerepelhet más, mint az = jel és egy karakterláncot visszaadó kifejezés és hogy legalább ezeknek egy sorban kell szerepelniük.

Itt álljunk meg egy pillanatra és vizsgáljuk meg az állomány elnevezését!

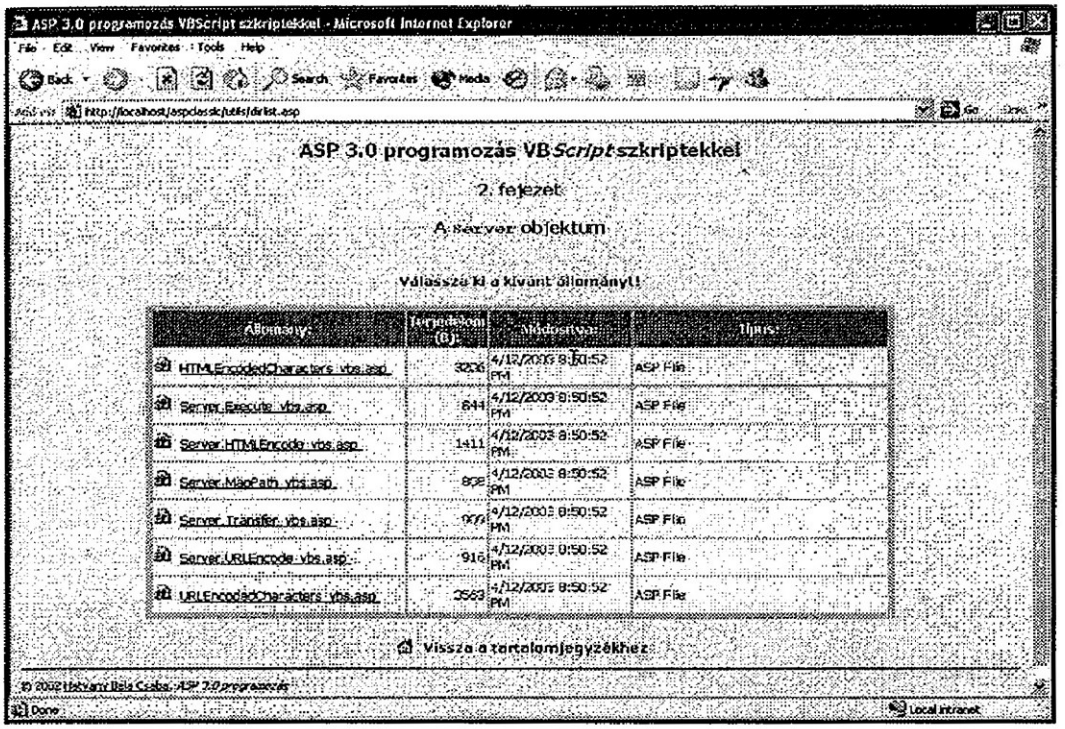
A `Server.Execute.asp` állományt ugyanúgy neveztem el, mint ahogyan a `Server` objektum `Execute` metódusát meg kell hívni, azaz `Objektum.Metódus`. Amennyiben lehetséges, az objektumok metódusainak és tulajdonságainak bemutatásakor betartom ezt a szabályt. Ez egyrészt megkönnyíti a példák keresését, másrészt egy kis rendet teremt a sok példa között. A példákat ebben a fejezetben objektumok szerint mappákba csoportosítottam. A `Server` objektumra vonatkozó példák az `/ASPClassic/VBS/2. fejezet/Server` mappában találhatók.

Most nézzük meg az előbb idézett kódrészletet! Az `Execute` metódus paramétere egy lap relatív URL-je legyen. Esetünkben egy HTML lapról van szó: `Server.Execute/testExecute.htm`. Ez a lap a példánk mappájába elhelyezett `Server.Execute` almappába van és a mappa neve utal a példa nevére. A `Server.Execute` mappa elkülöníti a `testExecute.htm` segédállományt a `Server` mappában levő többi példától. Ez a módszer megkönnyíti az példaállományok kiválasztását az MMC konzolban (2.1. ábra).



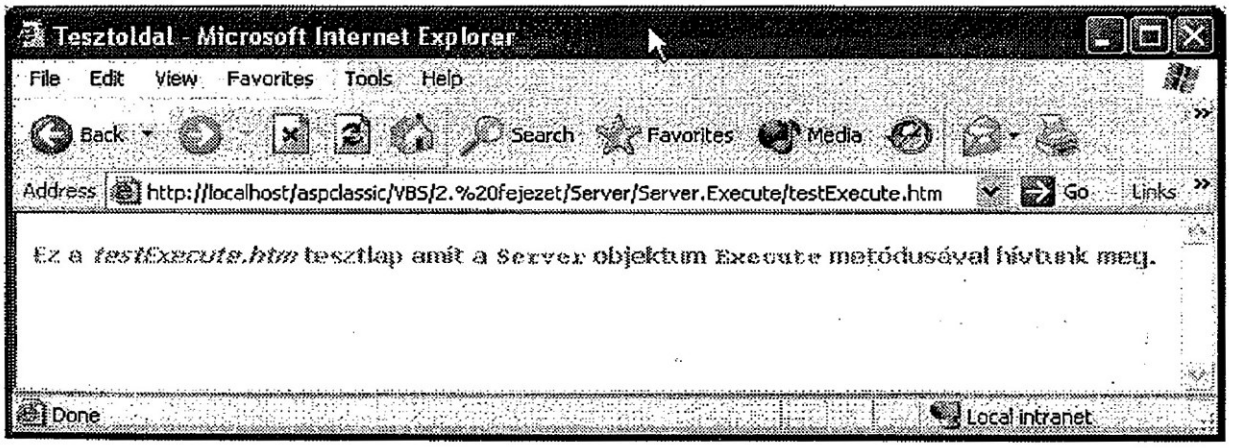
2.1. ábra A Server objektumra vonatkozó példák az MMC konzolban

Ha a példákat az `ASPClassic` webalkalmazás kezdőlapjáról közelítjük meg, akkor csak a példákat látjuk, a segédállományokat tartalmazó almappákat nem látjuk (2.2. ábra).



2.2. ábra A Server objektumra vonatkozó példák böngészőben

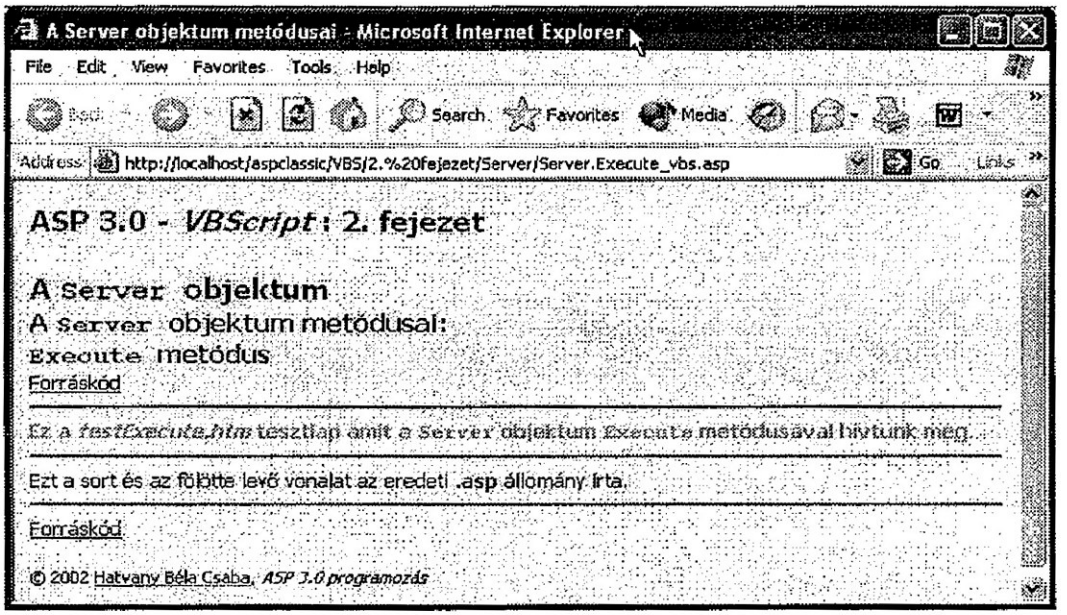
Most nézzük meg a `testExecute.htm` lapot a böngészőben. (2.3. ábra)!



2.3. ábra A testExecute.htm lap

Végül töltsük le a `Server.Execute.asp` lapot! A lapot a böngészőben a 2.4. ábrán láthatjuk.

Az `Execute` metódussal az eredeti lap tartalma is átkerül az új lapra még akkor is, ha az új lap egy másik webalkalmazáshoz tartozik. A böngészőben mindvégig az eredeti lap címét látjuk.

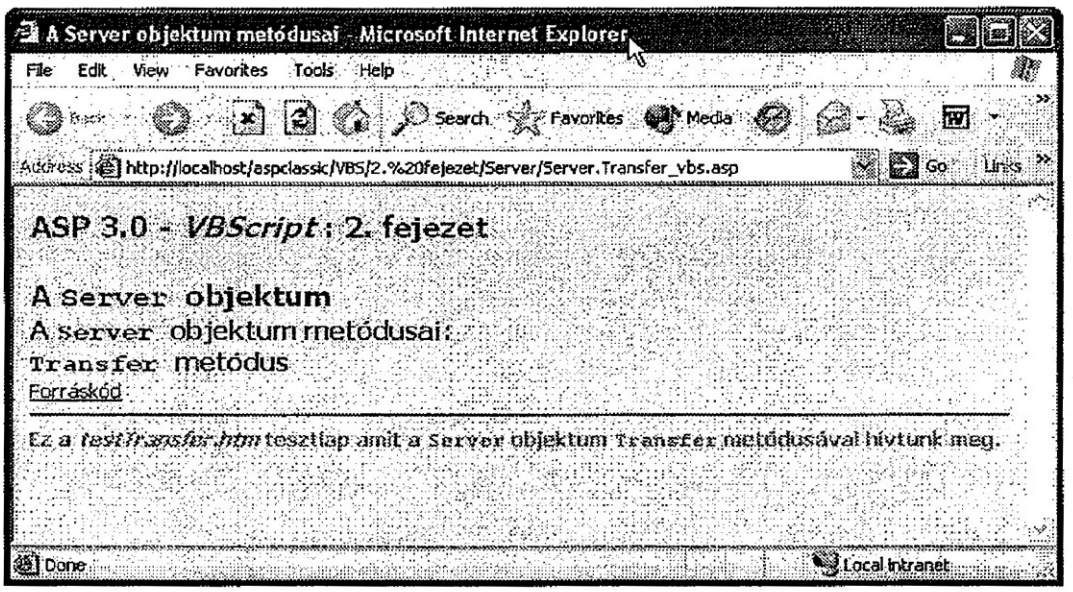


2.4. ábra A `Server.Execute` *metódus alkalmazása*

A `Server` objektum `Execute` metódusával rokon a `Transfer` metódus. Szintén egy ASP vagy egy HTML lap relatív URL-jét várja:

```
Server.Transfer url
```

A lap tartalma is átkerül az új lapra, pontosan, mint az `Execute` metódus esetében. A lényeges különbség a két metódus között abban áll, hogy a `Transfer` metódus esetében az új lap letöltése után az eredeti lap további utasításai nem lesznek végrehajtva. A `Transfer` metódus alkalmazását a `Server.Transfer.asp` lap illusztrálja. Az eredményt a 2.5. ábrán látjuk.



2.5. ábra A `Server.Transfer` *metódus alkalmazása*

A `Server` objektum `MapPath` metódusa egy ASP vagy HTML lap URL-jét várja, és az állomány fizikai elérési útját adja vissza:

```
strFilePath = Sever.MapPath url
```

Sok webalkalmazásban tudnunk kell egy bizonyos állomány fizikai elérési útját. Például a később bemutatásra kerülő `FileSystemObject` objektummal állományokat olvashatunk vagy írhatunk a szerveren, vagy meglévő állományok tulajdonságát kérdezhetjük le. Ilyenkor ismernünk kell a szóban forgó állomány fizikai elérési útját. Ebben az esetben a `MapPath` metódust használjuk, hogy az állomány helyét a szerveren megállapítsuk.

A `Server.MapPath_vbs.asp` állomány bemutatja a `MapPath` metódus használatát:

```
A jelenlegi állomány fizikai elérési útja: <br><br>
<%
Response.Write  Server.MapPath("Server.MapPath_vbs.asp")
%>
```

Az eredményt a 2.6. ábrán láthatjuk.

2.6. ábra A `MapPath` *metódus alkalmazása*

A `HTMLEncode` metódus a paraméternek átadott *unencoded* karakterláncot átülteti HTML kódba:

```
strEncoded = Server.HTMLEncode unencoded
```

Ha például

```
unencoded = "Így adunk ki egy szöveget dőlt " & _
"<FONT face=""Courier New""> <I>""Courier " & _
"New""</I></FONT> karakterkészlettel."
```

és ezt a szöveget a `Response.Write` metódussal kiírjuk egy ASP lapra, akkor az eredmény

> Így adunk ki egy szöveget dőlt *"Courier New"* karakterkészlettel.

A `Response.Write` metódus a böngészőnek továbbítja a karakterláncot és, mivelhogy ez a karakterlánc HTML kódot is tartalmaz, a böngésző értelmezi a kódot és megfelelően végrehajtja. Ha viszont nem az *undecoded* karakterlánc HTML értelmezését akarjuk a lapon megjeleníteni, hanem magát a karakterláncot, akkor minden olyan karaktert, aminek HTML értelmezése van, eleve helyettesítenünk kell a megfelelő HTML kóddal. Így például `<FONT>` helyett a `<FONT>` kódot kell írnunk és így tovább. Ebben segít a `HTMLEncode` metódus. Ha a `Response.Write` metódussal a `Server.HTMLEncode` *unencoded* által visszaadott karakterláncot küldjük a böngészőnek, akkor az eredmény

> Így írunk ki egy szöveget dőlt <FONT face="Courier New"><I>"Courier New"</I></FONT> karakterkészlettel.

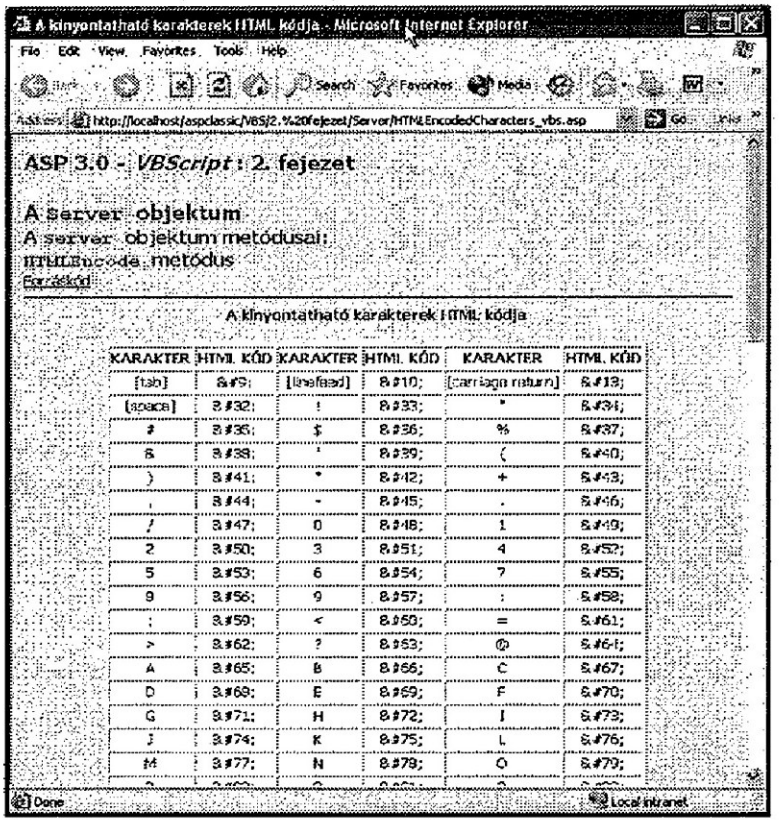
A fentieket a `Server.HTMLEncode_vbs.asp` állomány illusztrálja. Ha letöltjük ezt a lapot, meggyőződhetünk a `HTMLEncode` metódus működéséről: a HTML kódban nem engedélyezett karaktereket a metódus a megfelelő HTML kóddal helyettesíti. Ha a megjelenített lap forráskódját a böngészőből meghívott **NotePad**-del megnézzük, akkor az előbb idézett sor helyett ezt látjuk:

```
&#205;gy &#237;runk ki egy sz&#246;veget dolt &lt;FONT
face=&quot;Courier New&quot;&gt;&lt;I&gt;&quot;Courier
New&quot;&lt;/I&gt;&lt;/FONT&gt; karakterk&#233;szlettel.
```

Észrevehető, hogy `Server.HTMLEncode` metódus a HTML kódban szerepet játszó karaktereket, mint például <, >, vagy ", a megfelelő HTML kóddal helyettesíti, esetünkben sorra a `<`, `>`, illetve `"` kóddal. Az olyan karaktereket, melyeknek ANSI decimális kódja nagyobb, mint 126, szintén HTML kódba ülteti át. Így például az Í karaktert helyett a `Í` HTML kódot adja vissza. A `Response.Write` ezt a HTML kódra lefordított karakterláncot továbbítja a böngészőnek és ezzel magyarázható a megjelent szöveg.

Mint tudjuk, a kinyomtatható karakterek a HTML kódjukkal is megadhatók egy HTML lapban: az A karakter helyett, melynek a decimális ANSI kódja 65, a megfelelő `A` HTML kódot is használhatjuk. Ez a lehetőséget és a `Server.HTMLEncode`

metódus nyújtotta kódolást felhasználva, könnyen elkészíthetünk egy ASP lapot, ami egy táblázatba foglalja a kinyomtatható karakterek HTML kódját. Ezt mutatja be a `HTMLEncodedCharacters_vbs.asp` példánk. A lapot a 2.7. ábrán láthatjuk.



2.7 ábra A kinyomtatható karakterek HTML kódját a `Server.HTMLEncode` *metódussal jeleníthetjük meg*

A táblázat, ezúttal HTML lapként, a letöltött ZIP állományban is megtalálható. Elérési útja `Függelékek\html\HTMLEncodedCharacters.htm`, de elérhető a `Függelékek` mappában levő `index.htm` lapról is.

A `Server` objektum `URLEncode` metódusa átalakítja az „illegális" karaktereket URL kódra. Szintaxisa

```
strEncoded = Server.URLEncode unencoded
```

Gyakran előfordul, hogy egy HTML `<A HREF...></A>` elemben a szerverre küldünk *név* = *érték* párokat, mint a következő példában:

```
<%
dim strURL
strURL = "http://myserver.com/myasp.asp?" _
"title=ASP 3.0 programozás&author=Hatvany Béla Csaba"
Response.Write "<A HREF = " & strURL & _
               "> ASP 3.0 programozás </A>"
%>
```

Ilyen esetben a myasp.asp lapra átküldött *név = érték* párokat a lap URL-jéhez kell kötnünk, a legelső párt egy ? karakterrel, a többit egy & karakterrel. A HTML protokoll szerint szóközök nem engedélyezettek (mint esetünkben a title változónév ASP 3.0 programozás értékében vagy az author változónév Hatvany Béla Csaba értékében) és ugyancsak illegálisaknak számítanak a 126-nál nagyobb ANSI kóddal rendelkező karakterek (mint esetünkben a title változónév értékének az á karaktere). Bizonyos böngészők egy ilyen ASP lapkérést hibásan továbbítanak a szerverre. A példában szereplő lapkérést a **Netscape Navigator** így értelmezi:

```
http://myserver.com/myasp.asp?title=ASP
```

ami egy hibát okoz a szerveren. Ilyen és hasonló helyzeteket elkerülve játszik fontos szerepet az URLEncode metódus. Az előbbi kódrészlet helyesen:

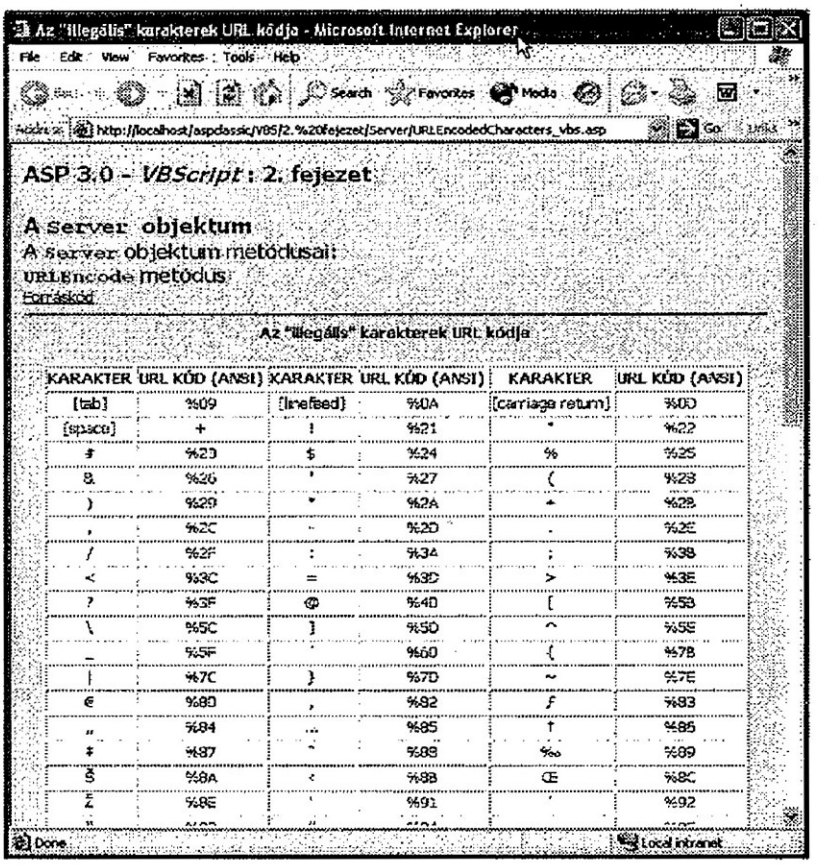
```
<%
Dim strURL
strURL = "http://myserver.com/myasp.asp?" _
"title=ASP 3.0 programozás&author=Hatvany Béla Csaba"
Response.Write "<A HREF = " & URLEncode (strURL) & _
"> ASP 3.0 programozás </A>"
%>
```

A URLEncode metódus minden, a HTML protokoll szerint, illegálisnak minősített karaktert az úgynevezett URL kódjával helyettesít, mely egy % karakterből és az illegális karakter hexadecimális ANSI kódjából áll. Az á karakternek, mivelhogy ANSI decimál kódja 225=14×16+1, az URL kódja %E1. A legális karaktereket az URLEncode metódus változatlanul hagyja.

Az URLEncode metódus felhasználásával egy táblázatba foglalhatjuk az összes illegális karaktert az URL kódjukkal együtt. Az URLEncodedCharacters_vbs.asp ASP lap létrehozza ezt a táblázatot, amit a 2.8. ábrán is láthatunk. A táblázat HTML formátumban a letölthető ZIP állományban is megtalálható. Elérési útja Függelékek\html\URLEncodedCharacters.htm, de elérhető a Függelékek mappában levő index.htm lapról is.

A Server objektum GetLastError metódusa lehetővé teszi, hogy részleteket kérjünk a szervertől a legutoljára bekövetkezett hibáról. Az ASP lapokon bekövetkezett hibák kezelésével az ASPError objektum leírásánál fogunk részletesen foglalkozni. Az GetLastError metódusának nincs paramétere. Visszatérési értéke egy referencia az ASPError objektumra:

```
<%
    Dim objASPError
    Set objASPError = Server.GetLastError()
%>
```



2.8. ábra A HTML protokoll szempontjából „illegálisnak" számító karakterek és URL kódjuk megjelenítése a `Server.URLEncode` *metódus segítségével*

2.2.2 A `ScriptTimeout` tulajdonság

A `Server` objektumnak csak egy tulajdonsága van, a `ServerTimeout` tulajdonság. Ezzel a tulajdonsággal lekérdezhetjük, illetve meghatározhatjuk az időt másodpercben, melynek letelte után az egy lapban levő szkriptek végrehajtása megszűnik:

```
<%
    Dim intTimeout
    Server.ScriptTimeout = 100
    intTimeout = Server.ScriptTimeout
%>
```

A `ScriptTimeout` tulajdonság alapértelmezett értéke 90 másodperc. A tulajdonság alkalmazásával elkerülhetjük, hogy esetleg hibás logikával programozott ASP lapok, például egy „végtelen" ciklus esetében, kisajátítsák a szerver erőforrásait: a `ScriptTimeout` által meghatározott idő letelte után a szerver kiüríti a hibás szkriptet a memóriájából és hibát jelent.

A `ScriptTimeout` tulajdonság alkalmazását a `Session` objektum tárgyalásánál fogjuk látni.

2.3 Az Application objektum

Miután egy webalkalmazást a webszerveren definiáltunk, azaz miután az alkalmazásnak megfelelő virtuális mappáját létrehoztuk, az ASP létrehoz egy objektumot, mely segítségével számon tartja az alkalmazást. Az Application objektumról van szó. Minden egyes webalkalmazásnak megfelel a saját Application objektuma és minden webalkalmazásnak csak egy Application objektuma van. Egy alkalmazáson belül az Application objektum bármelyik ügyfélgép által az alkalmazás bármelyik lapjáról elérhető és az alkalmazás mindegyik ügyfele ugyanazt az Application objektumot használja. Az Application objektum addig marad a szerver memóriájában, azaz addig létezik, ameddig nem állítjuk le a szervert vagy ameddig nem távolítjuk el az alkalmazást a szerverről.

Az Application objektum segítségével alkalmazásszintű változókat és objektumokat hozhatunk létre és kezelhetünk. Ezek a változók és objektumok az alkalmazás minden ügyfele által az alkalmazás bármelyik lapjáról elérhetők.

Az Application objektumnak két eseménye van: OnStart és OnEnd, két gyűjteménye: Contents és StaticObjects, két metódusa: Lock és UnLock és egy tulajdonsága: Value.

2.3.1 Az Application objektum eseményei

Az Application objektum OnStart eseménye akkor következik be, amikor az első kliens lehívja az alkalmazás egyik lapját, rendszerint a kezdőlapját. Természetszerűen az OnEnd esemény akkor következik be, amikor az alkalmazás megszűnik, tehát amikor leállítjuk a szervert vagy eltávolítjuk az alkalmazást a szerverről. E két eseménynek megfelel egy-egy eseménykezelő eljárás, az Application_OnStart és az Application_OnEnd. Ezekkel az eljárásokkal a global.asa állomány leírásakor találkoztunk. Most közelebbről megvizsgáljuk szerepüket. Legelőször jegyezzük meg, hogy a két eseménykezelő eljárást csak a global.asa állományban definiálhatjuk.

Az Application_OnStart eljárásban létrehozhatunk, és értékekkel elláthatunk alkalmazásszintű változókat, tömböket vagy objektumokat.

Az előző szakaszban látott global.asa állományt alakítsuk át a következőképpen:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server >

' a webalkalmazás indulásakor
Sub Application_OnStart()
  ' definiálunk egy változót
```

```
Dim Variable

Variable="Ez egy változó, amit a " & _
         "global.asa " & _
         "állományban definiáltunk és az " & _
         "Application " & _
         "objektumba egy alkalmazásszinten" & _
         " definiált változóba tároltuk."

' a Variable változót az appVariable
' alkalmazásszintű változóba tároljuk;
' az appVariable változó értékét az alkalmazás
' folyamán az alkalmazás
' bármelyik lapjáról elérhetjük
Application("appVariable")=Variable

' definiálunk egy tömböt
Dim varArray(4)

varArray(0) = "Ez egy tömb, amit a "
varArray(1) = "global.asa "
varArray(2) = "állományban definiáltunk és az "
varArray(3) = "Application "
varArray(4) = "objektumba egy alkalmazásszinten" & _
" definiált tömbbe tároltuk."
' a varArray tömböt az appArray alkalmazásszintű
' tömbbe tároljuk;
' az appArray tömb értékét az alkalmazás folyamán
' az alkalmazás
' bármelyik lapjáról elérhetjük
Application("appArray")=varArray

End Sub

  ' a webalkalmazás befejezésekor
Sub Application_OnEnd
  '
End Sub

</SCRIPT>
```

A fenti forráskód a letölthető ZIP állományban a `2. fejezet/Application` mappában elhelyezett `global1.asa` állományban található. Nézzük meg közelebbről! A szkript tartalmazza a két eseménykezelő eljárást, de csak az `Application_OnStart` eljárásba helyeztünk el kódot. Legelőször definiáltuk a `Variable` változót és hozzárendeltünk egy értéket. Ez egy lokális változó, ami csak az eljárásban létezik és mihelyt az eljárás végre lett hajtva a változó ki lesz ürítve a memóriából. A következő sorban létrehoztunk egy globális változót, pontosabban egy alkalmazásszintű változót, melynek neve `appVariable` és hozzárendeltük a `Variable` változó értékét. Az al-

kalmazásszintű változót az Application objektum Value tulajdonságának felhasználásával hozzuk létre és látjuk el egy értékkel:

```
Application.Value("alkalmazásszintű_változónév") = változónév
```

Mivelhogy az Application objektum alapértelmezett tulajdonsága a Value tulajdonság (más tulajdonsága nincs is), nem szükséges a tulajdonság explicit kiírása. Ezt a rövidített írásmódot használtam a szkriptben.

Ezután a varArray öt elemű tömböt definiáltuk és elemeit 0-tól 4-ig elláttuk értékekkel. Ez a tömbváltozó is lokális és megszűnik az eljárás elhagyásával, nem kell tovább törődnünk vele. A tömb értékét egy alkalmazásszintű, azaz globális tömbhöz rendeltük. Ezt is az Application objektum Value tulajdonságával hajthatjuk végre:

```
Application.Value("alkalmazásszintű_tömbnév") = tömbnév
```

Itt is elhagyhatjuk az alapértelmezett Value tulajdonságot.

Mivelhogy az eljárásban definiált Variable és varArray változók az eljárás végrehajtása után megszűnnek létezni, az alkalmazásszintű változók viszont megmaradnak az alkalmazás folyamán, bármikor lekérdezhetjük, illetve megváltoztathatjuk. Ezt egy ASP állományba írt szkripttel tehetjük meg. Hozzunk létre egy ASP állományt és írjuk bele a következő kódrészletet:

```
<%
  Response.Write Application("appVariable") & "<br>"
  Response.Write Application("appArray")(0)
  Response.Write "<b>" & _
                 Application("appArray")(1) & "</b>"
  Response.Write Application("appArray")(2) & "<br>"
  Response.Write "<b><FONT face='Courier New'>" & _
                 Application("appArray")(3) & _
                 "</FONT></b>"
  Response.Write Application("appArray")(4)
%>
```

Egy alkalmazásszintű változó értékét az Application objektum Value tulajdonsága adja vissza. Egy egyszerű változó esetében

```
változónév = _
Application.Value("alkalmazásszintű_változónév")
```

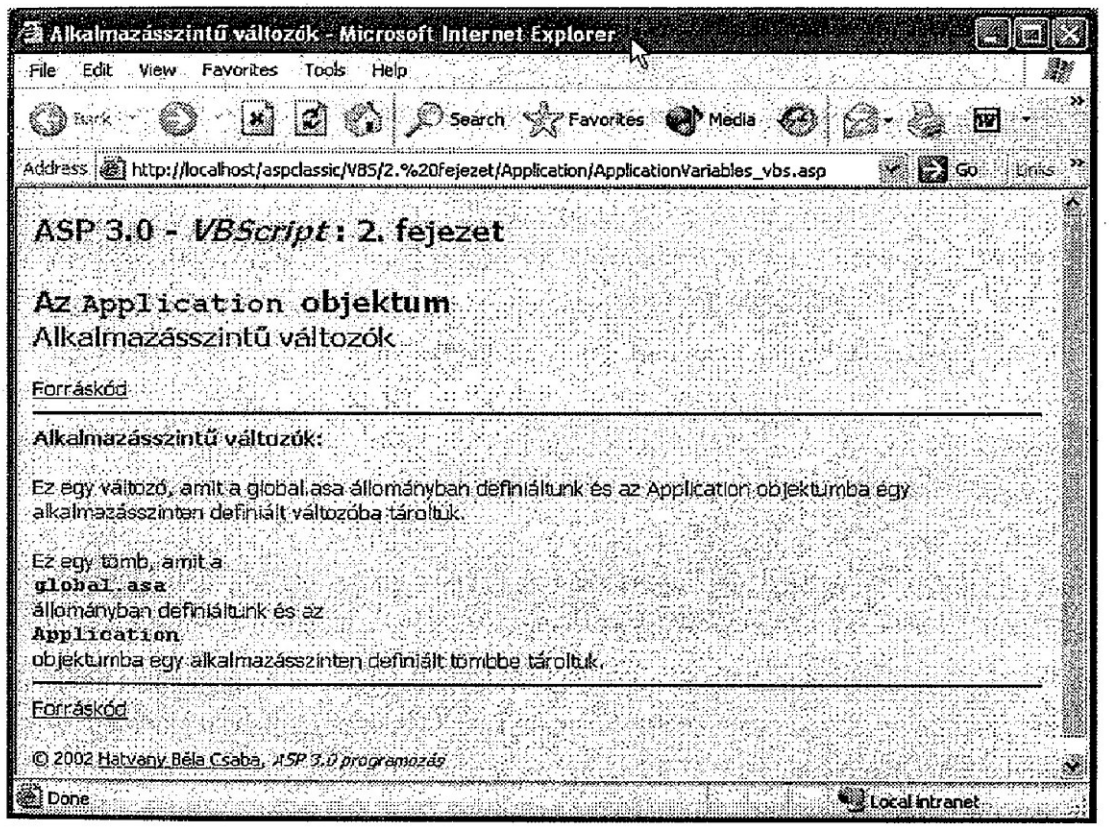
míg egy tömbváltozó esetében

```
tömbnév = _
Application.Value("alkalmazásszintű_tömbnév")
```

Itt is elhagyhatjuk az alapértelmezett Value tulajdonságot.

Mentsük el az állományt `ApplicationVariables_vbs.asp` név alatt az `Application` mappába. Mielőtt azonban ezt az állományt letöltenénk a böngészővel, még egy fontos tennivalónk van. Ugyanis a `global1.asa` állományt a webszerver nem ismeri fel: nem helyes a neve és rossz helyen van. Ezért másoljuk át a `glogal1.asa` állományt a `VBS` mappába `global.asa` név alatt. A `VBS` mappa egy webalkalmazás alapmappája és itt keresi a szerver a `global.asa` állományt. Most lehívhatjuk a böngészőnkkel az elmentett ASP állományt. Az eredményt a 2.9. ábrán láthatjuk.

Mikor a böngészőnkkel legelőször lekérjük a szervertől az ASP lapot, akkor a szerveren bekövetkezik az `OnStart` esemény. A szerver végrehajtja az `Application_OnStart` eljárást és ezzel létrehozza és értékeket rendel az eljárásban definiált alkalmazásszintű változókhoz. Ezután elkezdi a lekért lap előkészítését: értelmezi a lap ASP kódját, átalakítja HTML kóddá és visszaküldi a böngészőnek megjelenítésre. Ha másodszor kérjük le a lapot, akkor az `Application` objektum már működik, tehát az `OnStart` esemény ezúttal nem következik be, az eseménykezelő eljárás nem lesz végrehajtva, de a létrehozott alkalmazásszintű változók még mindig léteznek. Így a lekért lap megjelenítése minden nehézség nélkül megtörténhet.



2.9. ábra Alkalmazásszintű változók

Hasonlóképpen történnek a dolgok az alkalmazásszintű objektumokkal is. Módosítsuk a `global1.asa` állományt, úgy hogy a következő forráskódot tartalmazza és mentsük

el `global2.asa` néven, vagy tekintsük meg az `Application` mappában levő `global2.asa` állományt:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server >

' a webalkalmazás indulásakor
Sub Application_OnStart()
  ' definiálunk egy objektumot
  Dim objPageCounter

  Set objPageCounter=Server.CreateObject("MSWC.Counters")

  ' az objPageCounter objektumot az appObject
  ' alkalmazásszintű objektumba tároljuk;
  ' az appObject objektum metódusait az alkalmazás
  ' folyamán az alkalmazás
  ' bármelyik lapjáról elérhetjük
  Set Application("appObject") = objPageCounter
  ' az objPageCounter objektumra tovább
  ' nincs szükségünk,
  ' tehát törölhetjük
  Set objPageCounter = Nothing
  ' az utolsó négy utasítást egyszerűbben
  ' is végrehajthatjuk:
  ' Application("appObject") = _
    Server.CreateObject("MSWC.Counters")
  ' meghívjuk az objektum egyik metódusát
  Application("appObject").Set "intApplicationCounter", 0

End Sub

' a webalkalmazás befejezésekor
Sub Application_OnEnd
  ' töröljük az objektumot
  Set Application("appObject") = Nothing
End Sub

</SCRIPT>
```

Legelőször a `Server` objektum `CreateObject` metódusával egy egyszerű számoló-objektumot, az `MSWC.Counters` típusú `objPageCounter` objektumot hozzuk létre. Ez a számoló-objektum több számoló-változót tud egyszerre kezelni. Metódusai közé tartoznak a `Set` és az `Increment` metódusok. A `Set` metódussal egy számértéket rendelhetünk egy változóhoz:

```
objPageCounter.Set "változónév", számérték
```

Az `Increment` metódussal eggyel megnövelhetjük egy változó aktuális értékét:

```
objPageCounter.Increment "változónév"
```

Mivelhogy mi egy alkalmazásszintű objektumot akarunk létrehozni, definiálunk egy alkalmazásszintű objektumazonosítót és hozzárendeljük a létrehozott objektumot:

```
Set Application("appObject") = objPageCounter
```

Ezután a `objPageCounter` objektumra már nincs szükségünk és kiüríthetjük a memóriából. Végül az így létrehozott alkalmazásszintű számoló-objektummal az `intApplicationCounter` változóhoz hozzárendeljük a `0` értéket:

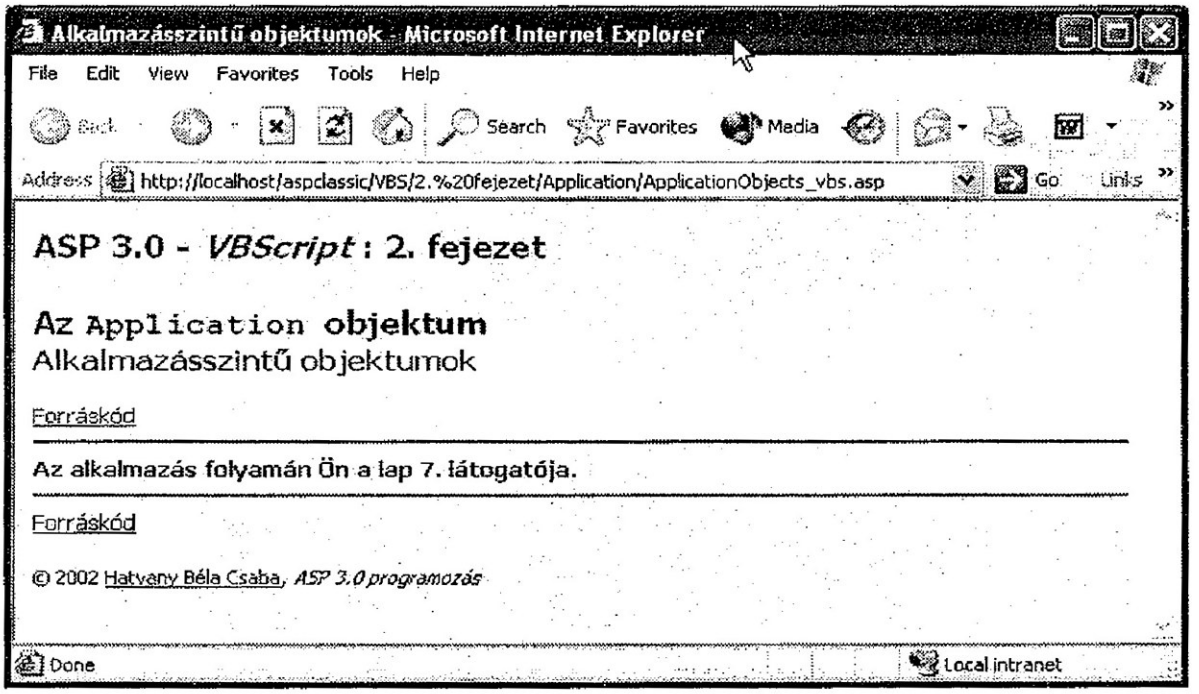
```
Application("appObject").Set "intApplicationCounter", 0
```

Az `intApplicationCounter` változó csak az `Application("appObject")` számoló-objektumon belül létezik. Ha az objektumon kívül hivatkozunk rá, akkor egy hibát idézünk elő (feltéve, hogy a szkript elején használtuk az `Option Explicit` utasítást). Az `Application("appObject")` objektum viszont az alkalmazás folyamán, bármely lapról, bármely klienstől elérhető, tehát az általa számon tartott `intApplicationCounter` változó is. Ez a lehetőséget felhasználva, számon tarthatjuk, hányszor látogatták meg az ügyfeleink az alkalmazásunkat.

Ezért létrehozzuk az `ApplicationObjects_vbs.asp` állományt, melynek a kódját az `Application` mappába megnézhetjük. Ebben az állományban az `Application("appObject")` alkalmazásszintű objektum által számon tartott `intApplicationCounter` változó értékét a lap letöltéskor eggyel megnöveljük.

Mielőtt a lapot a böngészőben megjelenítjük, maradjuk még egy keveset a `global2.asa` állománynál és vegyük észre, hogy az `Application_OnEnd` eljárásban az `Application("appObject")` objektumot megsemmisítettük, azaz kiürítettük a szerver memóriájából. Ha az alkalmazás leáll (nem azért mert senki sem hívja le a szerverről az alkalmazás lapjait – ettől még az alkalmazás fut – hanem azért mert az alkalmazást eltávolítottuk a szerverről vagy leállítottuk a szervert), akkor először az `Application_OnEnd` eljárás lesz végrehajtva és csak ezután távolítja el a szerver az alkalmazást a memóriájából. Ezért az `Application_OnEnd` eljárás a megfelelő hely, ahol szükségtelen objektumokat megsemmisíthetünk, vagy állományba menthetünk bizonyos adatokat, melyeket az alkalmazás újbóli indulásakor felhasználhatunk. Az alkalmazás megszűntekor az objektumai ugyan megszűnnek, azonban nem csak jó programozási stílusról vall, hanem a szerver zökkenőmentes működését is elősegítjük azzal, hogy az alkalmazás utolsó cselekedeteként megszűntetjük a felesleges objektumokat.

Most másoljuk a `global2.asa` állományt az VBS mappába levő `global.asa` állományra és hívjuk le a szerverről az `ApplicationObjects_vbs.asp` állományt (2.10. ábra)!



*2.10. ábra Egy alkalmazásszintű objektum (MSWC.*Counters*) számon tartja a látogatókat*

Ha a böngészőnket másodszor indítjuk, akkor a látogatók száma eggyel megnő, ha pedig a lapot a böngészőből frissítjük, akkor is eggyel megnő a látogatók száma, ugyanis a frissítéssel újra letöltjük a lapot. Ha a szervert leállítjuk, és újból elindítjuk, akkor a látogatók számolása elölről kezdődik.

2.3.2 Az `Application` objektum gyűjteményei

Az előző szakaszban láttuk, hogy a `Server` objektum `CreateObject` metódusával objektumokat tudunk létrehozni. Ha a `CreateObject` metódust a `global.asa` állományban az `Application_OnStart` eseménykezelő eljárásban hívjuk meg, akkor a létrehozott objektum egy alkalmazásszintű objektum. Ezen kívül van még egy lehetőségünk alkalmazásszintű objektumokat létrehozni.

A `global.asa` állományban az `<OBJECT>...</OBJECT>` HTML elemmel definiálhatunk egy alkalmazásszintű objektumot. Hozzunk létre egy újabb állományt `global3.asa` névvel és írjuk az állományba a következő kódrészletet:

```
<OBJECT ID="myAppCounter"
        RUNAT="Server"
        SCOPE="Application"
        PROGID="MSWC.Counters">
</OBJECT>
<OBJECT ID="myAppAdRotator"
        RUNAT="Server"
        SCOPE="Application"
        PROGID="MSWC.AdRotator">
```

```
</OBJECT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server >

' a webalkalmazás indulásakor
Sub Application_OnStart()
 Application.StaticObjects("myAppCounter").Set "Visitors", 0
End Sub

' a webalkalmazás befejezésekor
Sub Application_OnEnd
  ' töröljük az objektumot
 Set Application.StaticObjects("myAppCounter") = Nothing
End Sub

</SCRIPT>
```

Az állomány elején két objektumot definiálunk. Az `ID` attribútummal az objektum azonosítóját adjuk meg, a `RUNAT` attribútummal, melynek értéke `"Server"` vagy `"Client"` lehet, leszögezzük, hogy a szerveren kívánjuk az objektumot létrehózni a `SCOPE` attribútummal, melynek értéke `"Application"` vagy `"Session"` lehet, esetünkben úgy rendelkezünk, hogy alkalmazásszintű objektumokat hozzunk létre, a `PROGID` attribútummal pedig megnevezzük az objektum típusát. Az `<OBJECT>...</OBJECT>` HTML elem csak definiálja, de nem hozza létre az objektumokat. Az objektumok csak akkor jönnek létre, mikor egy utasítással legelőször rájuk hivatkozunk. Ezzel a módszerrel definiált objektumokat statikus objektumoknak nevezzük.

Az alkalmazásszintű statikus objektumokhoz az `Application` objektum `StaticObjects` gyűjteményével tudunk hozzáférni. A gyűjtemény a szokásos három tulajdonsággal, `Count`, `Item` és `Key`, valamint a szokásos két metódussal, `Remove` és `RemoveAll` rendelkezik.

A `Count` tulajdonság a gyűjtemény elemeinek számát adja vissza

```
intCount = Application.StaticObjects.Count
```

Az `Item` tulajdonság, mely egy paraméterrel rendelkezik és egyben az alapértelmezett tulajdonság, azaz nem kell feltétlenül kiírnunk, a paramétere által meghatározott objektumot adja vissza. A paraméter lehet az objektum (egyes alapú) sorszáma a gyűjteményben vagy az objektum neve, amit kulcsnak is nevezzünk.

```
obj = Application.StaticObjects.Item(1)
obj = Application.StaticObjects.Item(myAppCounter)
obj = Application.StaticObjects(1)
obj = Application.StaticObjects(myAppCounter)
```

A `Key` tulajdonság a gyűjtemény elemeinek kulcsát adja vissza. Egy paramétert vár, az objektum sorszámát a gyűjteményben:

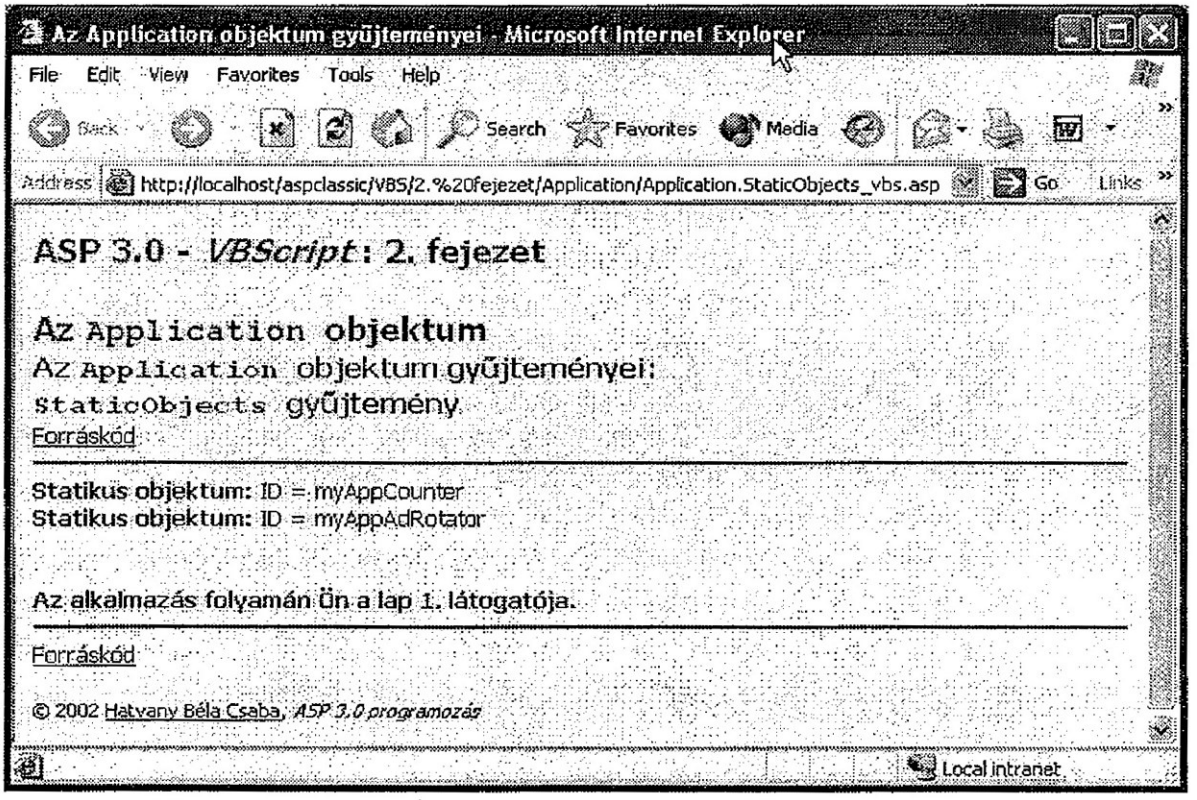
```
név = Application.StaticObjects.Key(sorszám)
```

Végül a két metódus eltávolít a gyűjteményből egy vagy az összes objektumot:

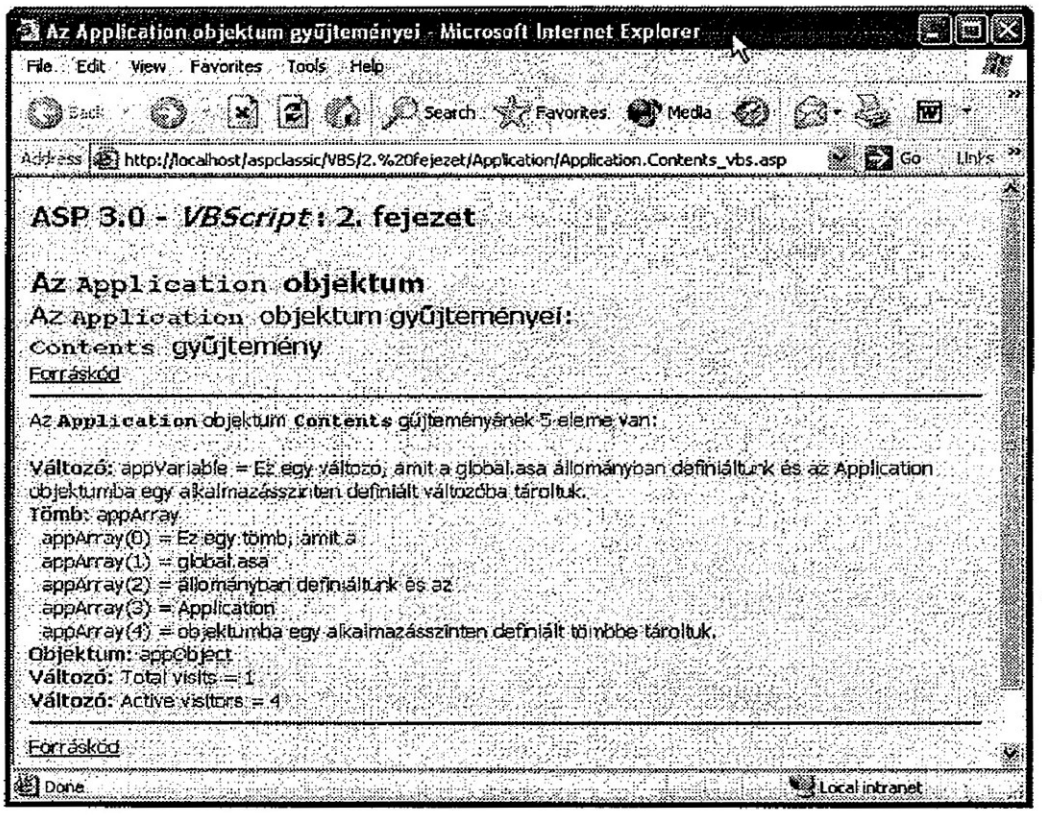
```
Application.StaticObjects.Remove(objektumnév)
Application.StaticObjects.RemoveAll()
```

Térjünk vissza a `global3.asa` állományhoz! Az `Application_OnStart` eljárásban meghívtuk az `MSWC.Counters` típusú számoló-objektum `Set` metódusát és ezzel létre is jött az objektum. Az `Application.StaticObjects_vbs.asp` ASP lap megmutatja, hogyan használhatjuk a `StaticObjects` gyűjteményt. Mielőtt a lapot letöltjük, ne felejtsük el a `global3.asa` állományt a `VBS` mappába `global.asa` névvel átmásolni (2.11. ábra).

Az `Application` objektum `Contents` gyűjteménye az összes alkalmazásszintű változókat és objektumokat tartalmazza. Csak azok az objektumok tartoznak ehhez a gyűjteményhez, melyeket a `Server.CreateObject` metódussal hoztunk létre. Az `<OBJECT>...</OBJECT>` HTML elemmel definiált statikus objektumok nem tartoznak a `Content` gyűjteményhez.



2.11. ábra Az `Application.StaticObjects` *gyűjtemény*



2.12. ábra Az `Application.Contents` *gyűjtemény*

Az `Application.Contents` gyűjtemény ugyanazokkal a tulajdonságokkal és metódusokkal rendelkezik, mint az `Application.StaticObjects` gyűjtemény: a `Count`, `Item` és `Key` tulajdonsággal, valamint a `Remove` és `RemoveAll` metódussal.

Az `Application.Contents_vbs.asp` állomány felsorolja `Contents` gyűjtemény tartalmát. Mielőtt az állományt letöltjük, másoljuk a mappájában levő `global4.asa` állományt a `VBS/global.asa` állományra. A `global4.asa` állomány magába foglalja az előbbi három `.asa` állományt, így biztosak lehetünk abban, hogy az alkalmazásban vannak alkalmazásszintű változók, objektumok és statikus objektumok. Az eredményt a 2.12. ábrán láthatjuk.

2.3.3 Az `Application` objektum metódusai

Az `Application` objektum két metódussal rendelkezik: `Lock` és `UnLock`. Mindkét metódus paraméternélküli metódus. A `Lock` metódus lezárja az `Application` objektumot minden más lap számára kivéve azt a lapot, amelyik a `Lock` metódust meghívta. Ne felejtsük el, hogy az alkalmazásszintű változók minden lapról bármely ügyfél által elérhetők! Fennáll tehát a veszély, hogy két vagy több ügyfél ugyanabban az időben ugyanazt a változót próbálja módosítani. Ez inkonzisztenciához vezethet. A `Lock` metódussal elkerülhetjük ezt a veszélyt: mielőtt egy alkalmazásszintű változót módosítanánk, meghívjuk az `Application.Lock` metódust. Ezzel hozzáférhetet-

lenné tesszük az Application objektumot a többi kliens részére. Miután módosítottuk az alkalmazásszintű változót, ismét hozzáférhetővé kell tegyük a többi ügyfél részére. Ezért mindjárt az alkalmazásszintű változó módosítása után meghívjuk az Application.UnLock metódust. Ha ezt elmulasztjuk, akkor a szerver a Server.ScriptTimeout idő letelte után teszi az Application objektumot ismét mindenki számára elérhetővé, ez azonban az alkalmazás teljesítményének a kárára megy. Ezért nagyon fontos, hogy csak addig zárjuk az Application objektumot, ameddig feltétlenül szükséges.

A Lock és UnLock metódus alkalmazását a Session objektum bemutatásánál látjuk majd akcióban.

2.4 A Session objektum

Az Application objektum az ASP hierarchikus objektum-struktúrában a webalkalmazásnak felel meg. A munkamenetnek ebben a struktúrában a Session objektum a megfelelője.

A Session objektumot az ASP akkor hozza létre, mikor egy ügyfél egy lapkéréssel bekapcsolódik a webalkalmazásba és addig létezik, ameddig az ügyfél nem hagyja el az alkalmazást. Itt pontosítani kell: mikor hagyja el az ügyfél az alkalmazást? Az ügyfél szempontjából ez akkor történik, amikor egy másik alkalmazás lapját tölti a böngészőbe, vagy amikor bezárja a böngészőt. A szerver szempontjából az ügyfél akkor hagyta el az alkalmazást, ha egy megadott időn belül, amit be lehet állítani, a szerver nem érzékel semmilyen aktivitást a kliens részéről. Ekkor a szerver megszünteti a kliens számára fenntartott Session objektumot. Ebből az következik, hogy csupán a böngésző nyitva tartásával nem biztosíthatjuk a Session objektum „végtelen" fennmaradását, illetve a böngésző bezárásával nem szűnik meg automatikusan a Session objektum. Minden ügyfél részére a szerver a Session objektum egy külön példányát tartja fenn, ezért a Session objektum alkalmas kliensorientált adatok kezelésére.

Az egyik lehetőséget az ilyen ügyfélorientált adatkezelésre a sütik (*cookies*) adják. A sütikről többet fogunk megtudni egy külön szakaszban. Most csak annyit jegyezzünk meg, hogy a szerver a Session objektum minden egyes létrehozott példányának egy egyértelmű munkamenet-azonosítót hoz létre és elküldi a böngészőnek, mely egy sütiben elhelyezi a kliensgépen a memóriába (feltéve, hogy a böngésző nem tiltja a sütik fogadását). A süti tartalmazza az alkalmazás alapmappájának virtuális elérési útját. Ameddig a kliens az alkalmazáson belül lapról lapra navigál, addig a szerver a süti segítségével számon tarthatja és ugyanazt a Session objektumot rendeli hozzá. Ha a kliens böngészője nem fogad sütiket, akkor minden új lap letöltésével a kliens egy új Session objektumot kap a szervertől. Ezek a sütik nem szerepelnek a

Response és Request objektumok Cookies gyűjteményében, amiről később még szó esik.

A Session objektum két eseménnyel rendelkezik:

OnStart
OnEnd

négy tulajdonsággal:

CodePage
LCID
Timeout
Value

két gyűjteménnyel:

Contents
SaticObject

és egy metódussal:

Abandon

A felsorolásból látszik, hogy a Session objektum sokban hasonlít az Application objektumra és valóban, az azonos nevű események, tulajdonságok és gyűjtemények hasonlóan is viselkednek.

2.4.1 A Session objektum eseményei

Az objektum két eseményét a Session_OnStart és a Session_OnEnd eljárás kezeli, amit az alkalmazás global.asa állományába kell definiálni.

Kezdjük máris egy példával! Nézzük meg a Session mappába levő global1.asa állományt:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server >

' a webalkalmazás indulásakor
Sub Application_OnStart()
 Application("Total visits")=0
 Application("Active visitors")=0
End Sub

' a munkamenet indulásakor
Sub Session_OnStart()
 Dim intTotalVisits

 Session.Timeout = 1
 Application.Lock
```

```
    Application("Total visits") = _
                Application("Total visits ") + 1
    intTotalVisits = Application("Total visits ")
    Application.UnLock
    Session("Total visits") = intTotalVisits
    Application.Lock
    Application("Active visitors") = _
                Application("Active visitors") +1
    Application.UnLock
End Sub

' a munkamenet befejezésekor
Sub Session_OnEnd
    Application.Lock
    Application("Active visitors") = _
                Application("Active visitors") -1
    Application.UnLock
End Sub

' a webalkalmazás befejezésekor
Sub Application_OnEnd

End Sub

</SCRIPT>
```

Az alkalmazás indulásakor két alkalmazásszintű változót hozunk létre és mindkettőnek a 0 értéket adjuk. Az `Application("Total visits")` változóval a látogatások számát, az `Application("Active visitors")` változóval a pillanatnyilag aktív látogatók számát kívánjuk követni.

Amikor egy munkamenet elkezdődik, a szerver létrehoz egy `Server` objektumot és így bekövetkezik az újonnan létrehozott `Session` objektum `OnStart` eseménye. A `Session_OnStart` eseménykezelő eljárásban legelőször a `Session.Timeout` tulajdonságnak adunk egy értéket.

A `Session` objektum `Timeout` tulajdonsága megszabja, mennyi időn belül szünteti meg a szerver a `Session` objektumot, ha a szerver nem érzékel semmilyen aktivitást a kliens részéről. A tulajdonság értékei egész számok és perceket jelentenek. Ha tehát `Session.Timeout = 5`, akkor öt perc után, ha a szerver nem kapott semmilyen kérést a klienstől, megszünteti a kliens részére létrehozott `Session` objektumot. A kliens böngészője továbbra is tartalmazza az alkalmazás lapját, a kapcsolat a szerverrel sem szűnik meg, a munkamenetszintű változók viszont, melyek a munkamenet alatt lettek létrehozva, elvesznek.

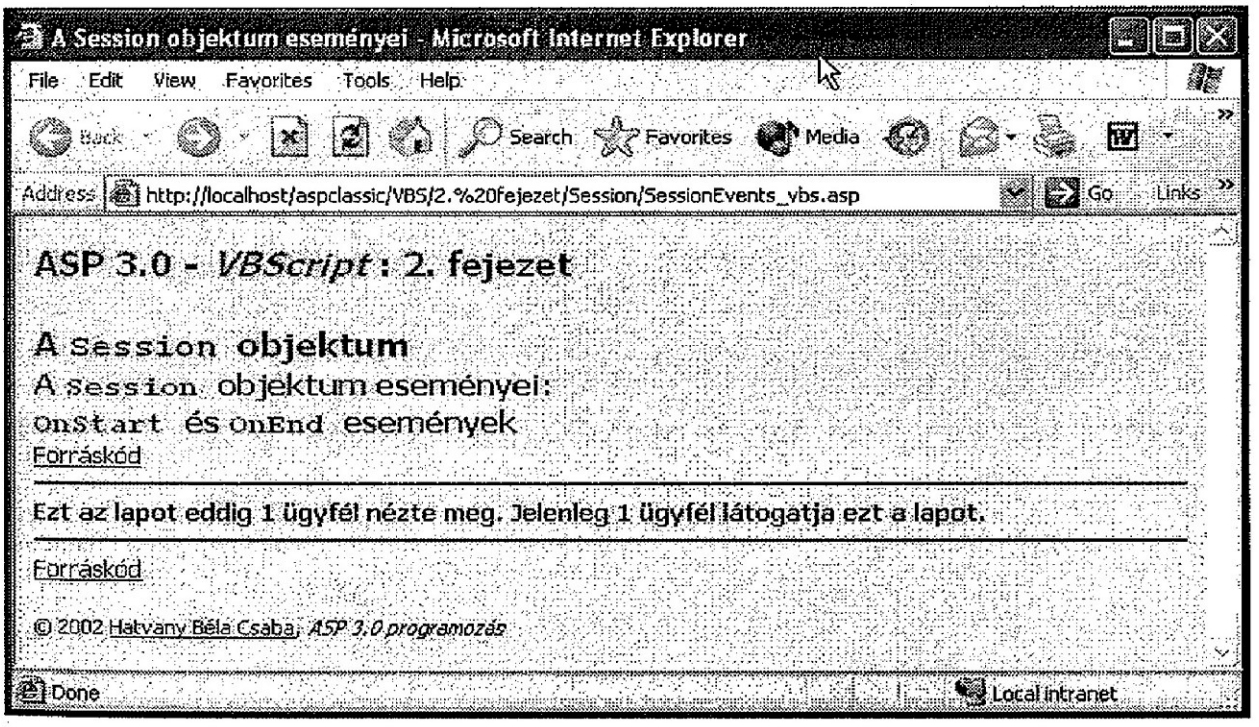
A `Timeout` tulajdonság komoly gondot okozhat, ha értéke nincs megfelelően meghatározva. Ha az időtartam túl rövid, akkor például az ügyfél nem lesz képes időben kitölteni egy űrlapot és nem lesz módjában a kért adatokat a szerverre felküldeni. Ha az időtartam túl hosszú, akkor bizonyos esetekben a szerver teljesítőképessége szen-

ved kárt. Ha például munkamenet szintű változókba adatbázis lekérdezésből származó rekordhalmazokat tárolunk és lapunkat sok ügyfél látogatja egyszerre, akkor a munkamenet szintű változók a szerver memóriájának jelentős részét egy hosszú időre lefoglalják. A `Timeout` tulajdonság alapértelmezett értéke 10 perc. Egy jól bevált érték tapasztalat szerint 20 perc.

Példánkban a munkamenet maximális időtartamát egy percre tettük. Ez az érték a reális alkalmazás esetén túl rövid, a mi példánkban a munkamenetet olyan rövidre kívánjuk szabni, amennyire csak lehet.

Miután rögzítettük egy munkamenet maximális időtartamát, alkalmazásszintű változókat készülünk módosítani. Ezért a módosítás idejére az `Application.Lock` metódussal lezárjuk az `Application` objektumot, hogy más kliens ne jusson az alkalmazásszintű változókhoz. Megnöveljük eggyel az `Application("Total visits")` és `Application("Active visitors")` változók értékét, majd ismét mindenki részére hozzáférhetővé tesszük az `Application` objektumot az `Application.UnLock` metódus meghívásával. Ezzel a `Session_OnStart` eljárás be is fejezi tevékenységét. Valahányszor ez az eljárás végre lesz hajtva, alkalmazásszintű változókkal, tehát olyan változókkal, melyek a munkamenet befejezése után is a szerveren maradnak, számon tartjuk a látogatások és az aktív látogatók számát.

Azért, hogy az aktív látogatók száma megfeleljen a valóságnak, egy munkamenet megszűntével az `Application("Active visitors")` változó számát eggyel csökkentenünk kell. Ez a `Session_OnEnd` eljárásban történik meg.



2.13. ábra A Session objektum eseményei számon tarthatják a látogatásokat és a látogatókat

Ez eredmény megtekintésére egy ASP lapot használunk. A `Session` mappában levő `SessionEvents_vbs.asp` állomány egyszerűen lekérdezi a változók értékét és megjeleníti az eredményt. Írjuk át a `VBS` mappában levő `global.asa` állományt a jelenlegi `global1.asa` állománnyal, majd töltsük le az említett `SessionEvents_vbs.asp` állományt a böngészőnkkel. Ha eddig nem hívtuk még le ezt a lapot, akkor az eredményt a 2.13. ábrán láthatjuk.

Indítsuk el a böngészőnket még egyszer (nem elég, ha a már megnyitott böngészőből egy új ablakot nyitunk!) és töltsük le ismét a lapot! A második böngészőben arról értesülünk, hogy lapunkat eddig kétszer nézték meg, és hogy jelenleg két látogatója van. Most gyorsan, a lap letöltése után még egy percen belül, csukjuk be a második böngészőt. Ez első böngészőben a szöveg változatlan: 1 látogatás és 1 látogató. Frissítsük a lapot: 2 látogatás, 2 látogató. Az első böngésző helyesen mutatja a látogatások számát, de még nem értesült arról, hogy közben a második látogató leköszönt, ugyanis a munkamenetre kiszabott maximális időtartam még nem járt le. Várjuk ki az egy perc elmúlását és frissítsük ismét a lapot az első böngészőben. Most az eredmény helyes: 2 látogatás, 1 látogató. Végezetül várjunk még egy percet, amíg biztos az első böngészővel elindított munkamenet is lejár. Természetesen közben nem szabad frissíteni a lapot. Az egy perc letelte után indítsuk újra a második böngészőt. A megjelenített szöveg tudatja velünk: 3 látogatás, 1 látogató. Hiába maradt nyitva az első böngésző, a munkamenete lejárt, tehát a jelentés helyes.

Vizsgáljuk meg a `Session` objektum tulajdonságait!

2.4.2 A `Session` objektum tulajdonságai

A `Timeout` tulajdonsággal már találkoztunk. A `Value` tulajdonságnak pontosan olyan szerepe van, mint az `Application` objektum esetében és a `Session` objektum alapértelmezett tulajdonsága. Más szóval, ha nem írjuk ki kifejezetten, melyik tulajdonsággal dolgozunk, akkor az ASP a `Value` tulajdonságot alkalmazza. Újak viszont a `SessionID`, `Codepage` és `LCID` tulajdonságok.

A `SessionID` tulajdonság az aktuális munkamenet egyértelmű azonosítószámát adja vissza. Értékét a szerver hozza létre, a webalkalmazások lekérdezhetik, de nem változtathatják meg az értékét:

```
StrID = Session.SessionID
```

A munkamenet azonosítóját a szerver a munkamenet létrehozásakor generálja és gondoskodik arról, hogy *az alkalmazáson belül* egyértelmű legyen. Nagyon fontos figyelembe venni, hogy a munkamenetek azonosítója csak egyazon alkalmazáson belül azonosítja egyértelműen a munkamenetet. Ezért értelmetlen, sőt az alkalmazás bizton-

ságát veszélyezteti a munkamenet-azonosítók tárolása, például egy adatbázisban. Ez a gyakorlat oda vezethet, hogy az egyik ügyfél hozzáférhet egy másik ügyfél bizalmas adataihoz. A munkamenet-azonosítókat viszont sikerrel használhatjuk az aktív ügyfelek megkülönböztetésére. Erre vonatkozó példát az `Abandon` metódus tárgyalásakor fogunk látni.

A `CodePage` tulajdonsággal lekérdezhetjük, illetve megadhatjuk a használt karakterkészletet egy munkameneten belül. Mint tudjuk, egy lap karakterkészletét a lap első sorában a

```
<%@ CODEPAGE="code_page_number" %>
```

végrehajtási rendelkezéssel is megadhatjuk, ahol *code_page_number* a kívánt karakterkészlet száma. Ha a munkamenet karakterkészletét egy lapon a

```
Session.CodePage = code_page_number
```

utasítással adjuk meg, akkor addig, ameddig másképp nem rendelkezünk, a megadott karakterkészlet az egész alkalmazásra érvényes. Ha ezenkívül egy lapon a végrehajtási rendelkezéssel is egy karakterkészletet meghatározunk, akkor ez a rendelkezés átírja a munkamenet részére megadott karakterkészletet, de az átírás csak a szóban forgó lapra érvényes. Ha ugyanezen a lapon ismét a `Session` objektum `CodePage` tulajdonságával megadunk egy újabb karakterkészletet, akkor ez az utolsó utasítás megint az egész alkalmazásra érvényes, de csak addig ameddig másképpen nem rendelkezünk.

Magyar nyelvű lapok esetében tanácsos az `1250` (*Central European Alphabet (Windows)*) vagy `28592` (*Central European Alphabet (ISO)*) kódszámú karakterkészleteket használni. A használatos karakterkészletek kódja a letölthető ZIP állományban a `Függelékek` mappában megtalálható.

Az `LCID` tulajdonság, a helyi azonosító (*locale identifier*) és a helyi dátum és pénznem sajátosságokat rögzít. A `CodePage` tulajdonsághoz hasonlóan, az `LCID` is a `Session` tulajdonságaként vagy egy végrehajtási rendelkezéssel adható és két lehetőség között ugyanaz az elsőbbségi szabály érvényes, mint a `CodePage` esetében. Például az Egyesült Államok helyi azonosítója `1033` (decimális) vagy `&H0409` (hexadecimális), Magyarországé `1038` vagy `&H040E` és Portugáliáé `2070` vagy `&H0816`. Az alábbi kódrészlet:

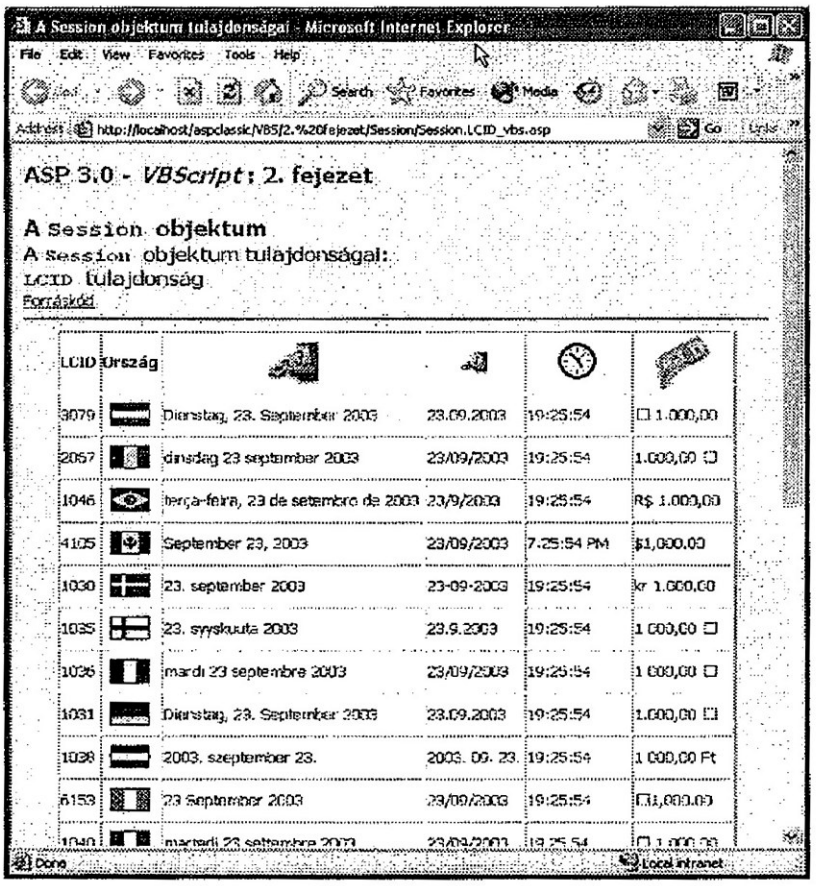
```
<%
 Response.Write "United Sates:"& "<br>"
 Session.LCID=&H0409
 Response.Write FormatDateTime(Now,1) & "<br>"
 Response.Write FormatDateTime(Now,2) & "<br>"
 Response.Write FormatCurrency(1000) & "<br>"
 Response.Write "Magyarország:"& "<br>"
 Session.LCID=&H040e
```

```
  Response.Write FormatDateTime(Now,1) & "<br>"
  Response.Write FormatDateTime(Now,2) & "<br>"
  Response.Write FormatCurrency(1000) & "<br>"
  Response.Write "Portugal:"& "<br>"
  Session.LCID=&H0816
  Response.Write FormatDateTime(Now,1) & "<br>"
  Response.Write FormatDateTime(Now,2) & "<br>"
  Response.Write FormatCurrency(1000) & "<br>"
%>
```

a következőt fogja eredményezni:

```
United Sates:
Saturday, February 02, 2002
02/02/2002
$1,000.00
Magyarország:
2002. február 2.
2002. 02. 02.
1.000,00 Ft
Portugal:
sábado, 2 de Fevereiro de 2002
02-02-2002
1.000$00 Esc.
```

(Az ASP 3.0 verziója még ismeri az Európai Unió közös új pénznemét.)



2.14. ábra A `Session.LCID_vbs.asp` *lap megmutatja a szerveren definiált helyi azonosítókat*

A 2.14. ábrán látható `Session.LCID_vbs.asp` lappal ellenőrizhetjük, melyik helyi azonosítót ismer a szerverünk.

A lap VBScript kódjában az fent idézett `FormatDateTime` és `FormatCurrency` függvényeket használtam. Mivelhogy JScript nem rendelkezik megfelelő függvényekkel, a `Session.LCID_vbs.asp` lapnak nincs JScript megfelelője.

2.4.3 A `Session` objektum gyűjteményei

Munkamenetszintű változókat és objektumokat az alkalmazás `global.asa` állományban elhelyezett `Session_OnStart` eljárásban hozhatunk létre, illetve az objektumok esetében, a `Session_OnEnd` eljárásban szüntethetünk meg. A következő `global2.asa` állományban a `Session("sesVariable")` változót és a `Session("sesArray")` tömböt munkamenetszinten hozzuk létre:

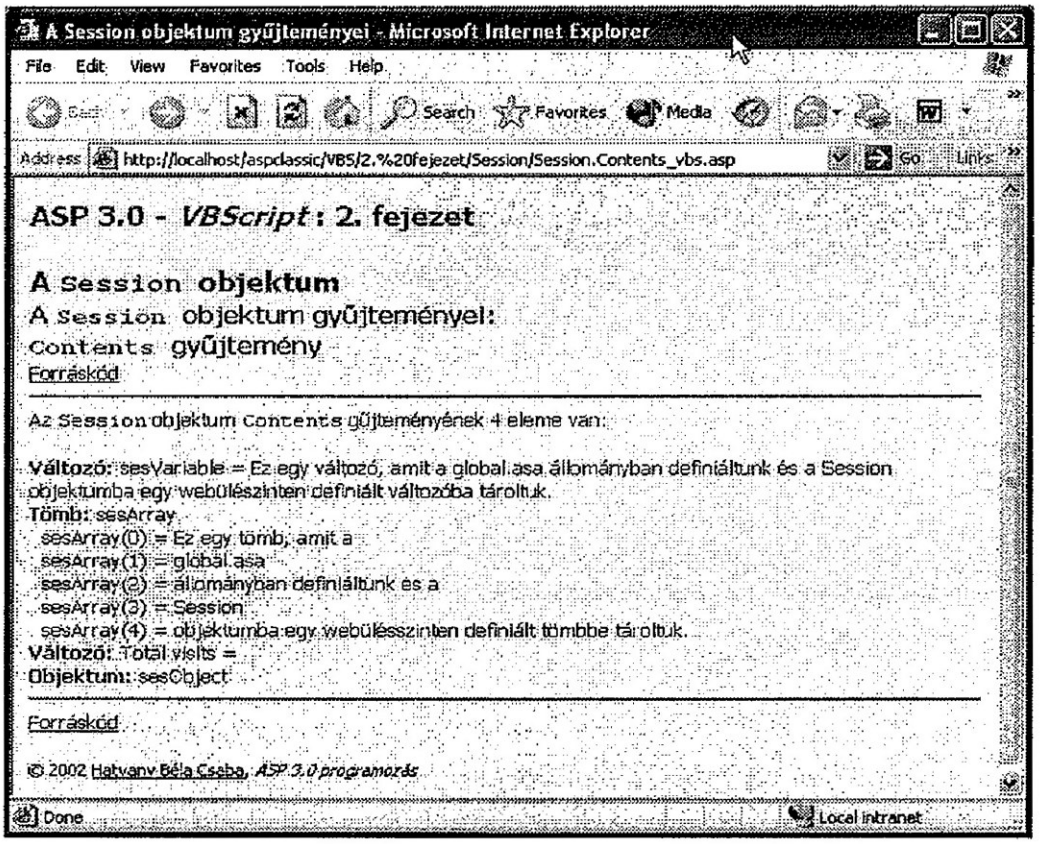
```
<SCRIPT LANGUAGE=VBScript RUNAT=Server >
' a munkamenet indulásakor
Sub Session_OnStart()
    ' definiálunk egy változót
    Dim Variable
    Variable = "Ez egy változó, amit a " & _
     "global.asa állományban definiáltunk és " & _
     "a Session objektumba egy munkamenetszinten " & _
     "definiált változóba tároltuk."
    ' a Variable változót az sesVariable
    ' munkamenetszintu változóba tároljuk;
    ' az sesVariable változó értékét a munkamenet
    ' folyamán az alkalmazás bármelyik lapjáról
    ' elérhetjük
    Session("sesVariable") = Variable
    ' definiálunk egy tömböt
    Dim varArray(4)
    varArray(0) = "Ez egy tömb, amit a "
    varArray(1) = "global.asa "
    varArray(2) = "állományban definiáltunk és a "
    varArray(3) = "Session "
    varArray(4) = "objektumba egy " & _
    " munkamenetszinten definiált tömbbe tároltuk. "
    ' a varArray tömböt az sesArray
    '  munkamenetszintű tömbbe tároljuk;
    ' az sesArray tömb értékét a munkamenet folyamán
    '  az alkalmazás
    ' bármelyik lapjáról elérhetjük
    Session("sesArray") = varArray
End Sub

</SCRIPT>
```

A global3.asa állományban a munkamenet szintű Session("sesObject") objektumot hozzuk létre, illetve a Session_OnEnd eljárásban töröljük:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server >
' a munkamenet indulásakor
Sub Session_OnStart()
        ' definiálunk egy objektumot

        Dim objPageCounter
        Set objPageCounter = _
             Server.CreateObject("MSWC.Counters")
        ' az objPageCounter objektumot az sesObject
        ' munkamenetszintű
        ' objektumba tároljuk;
        ' az sesObject objektum metódusait a munkamenet
        ' folyamán az
        ' alkalmazás bármelyik lapjáról elérhetjük
        Set Session("sesObject") = objPageCounter
        ' az objPageCounter objektumra tovább nincs
        ' szükségünk,
        ' tehát törölhetjük
        Set objPageCounter = Nothing
        ' az utolsó négy utasítást egyszerűbben is
        ' végrehajthatjuk:
        ' Session("sesObject") = _
        ' Server.CreateObject("MSWC.Counters")
        ' meghívjuk az objektum egyik metódusát
        Session("sesObject").Set "SessionCounter", 0
End Sub

' a munkamenet befejezésekor
Sub Session_OnEnd
        ' töröljük az objektumot
        Set Session("sesObject") = Nothing
End Sub
</SCRIPT>
```

A munkamenetszintű változók és objektumok használatát a SessionVariables_vbs.asp, illetve a SessionObjects_vbs.asp lapok mutatják be. Az első lap sikeres letöltéséhez másoljuk a global2.asa állományt az alkalmazás global.asa állományára, a második lap esetében a global.asa legalább a global3.asa kódját kell tartalmazza.

A Session objektum két gyűjteménye, StaticObjects és Contents mindenben megegyeznek az Application ugyanilyen nevű gyűjteményeivel. A különbség csupán annyiban áll, hogy a Session.StaticObjects csak a munkamenetszintű statikus objektumok érhetők el és hasonlóképpen a Session.Contents csak munkamenetszintű változók és (nem statikus) objektumok érhetők el. A Session.StaticObjects_vbs.asp és a Session.Content_vbs.asp példalapok ezeknek a gyűjteményeknek a használatát mutatják be.



2.15. ábra A `Session.Contents` *gyűjtemény alkalmazása*

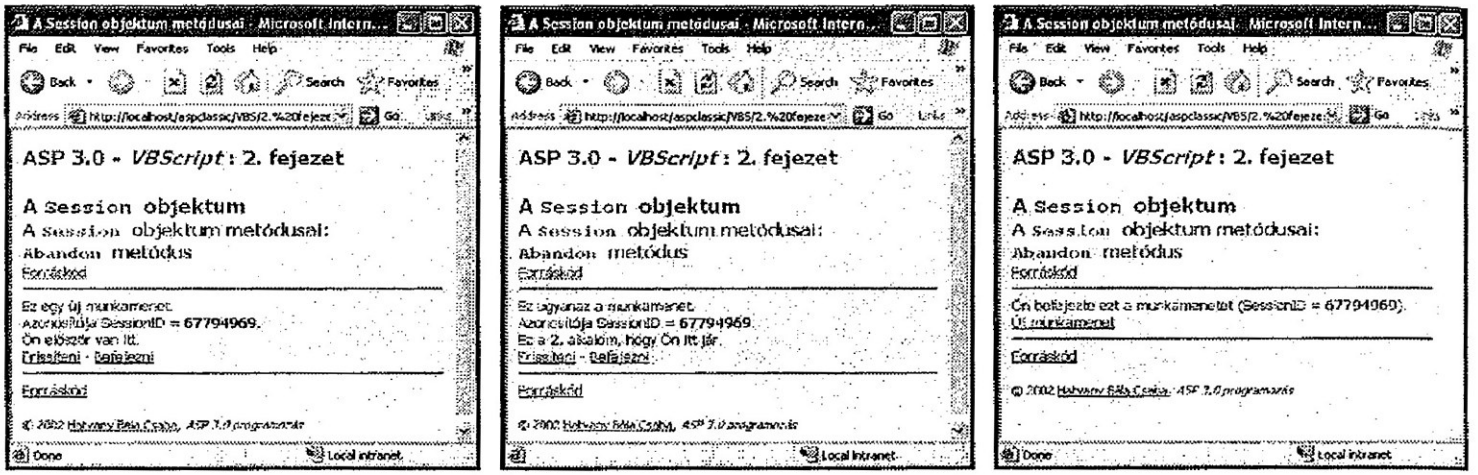
Mielőtt ezeket a lapokat letöltenénk, módosítsuk a `global.asa` állományt, illetve másoljuk rá a mappájukban található `global4.asa` állományt. Ez az állomány az előbbi `global1.asa`, `global2.asa`, és `global3.asa` kódját egyesíti és ha az alkalmazás `global.asa` állomány ezt a kódot tartalmazza, akkor a `2. fejezet\Session` összes példalapja sikeresen letölthető. A `Session.Content` gyűjtemény segítségével lekérdezett munkamenetszintű változók és objektumok listáját 2.15. ábrán láthatjuk.

2.4.4 A `Session` objektum `Abandon` metódusa

A Session objektum tárgyalását az `Abandon` metódus bemutatásával zárjuk. A `Session.Abandon` metódus meghívásakor a szerver befejezi az aktuális munkamenetet, miután megjelenítette a lapot, azaz miután elküldte a választ az ügyfél kérésére. A munkamenet megszűnte után a szerver kiüríti a memóriájából a megszűnt munkamenethez tartozó összes webszintű változót és objektumot. Ha ezután az ügyfél egy újabb lapot kér a szervertől a szerver egy új munkamenetet nyit az ügyfél részére.

Az `Abandon` metódus használatát a `Session.SessionID_vbs.asp` és `Session.Abandon_vbs.asp` lapok mutatják be. A forráskód egyúttal megmutatja a `Session` objektum `SessionID` tulajdonságának használatát és láthatjuk, hogy mun-

kamenetszintű változókat a `global.asa` állományon kívül is létrehozhatunk, illetve módosíthatunk.



2.16. ábra A `Session` *objektum* Abandon *metódusa és a* `SessionID` *tulajdonsága*

Legelőször töltsük le a `Session.SessionID_vbs.asp` lapot (2.16. ábra baloldali kép). A lapon megjelenik az aktuális munkamenet azonosítója és egy szöveg, mely értesít, hogy a jelenlegi munkamenet alatt most látogatjuk először ezt a lapot. Ha a lapot a böngésző menüjéből vagy a lapon levő Frissíteni kapcsolattal frissítjük, akkor látni fogjuk, hogy a munkamenet-azonosító nem változik, de a lapon megjelenített szöveg most tudtunkra hozza, hányadszor látogattuk meg a lapot a jelenlegi munkamenet folyamán (2.16 ábra középső kép). A lap forráskódja a következő:

```
<%
    If Session("counter") < 1 Then
        Session("counter") = 1
       Response.Write("Ez egy új munkamenet.<br>" & _
         " Azonosítója SessionID = <b>" & _
         Session.SessionID & "</b>.<br> Ön először van itt.<br>")
    Else
        Session("counter") = Session("counter") + 1
        Response.Write("Ez ugyanaz a munkamenet." & _
         "<br> Azonosítója SessionID = <b>" & _
         Session.SessionID & "</b>.<br> Ez a <b>" & _
         Session("counter") & _
         ".</b> alkalom, hogy Ön itt jár.<br>")
    End If

%>

<a href="Session.SessionID_vbs.asp">Frissíteni</a> -
<a href="Session.Abandon_vbs.asp">Befejezni<a>
```

Amikor ezt a kódot a szerver legelőször végrehajtja és a `Session("counter") < 1` kifejezést kiértékeli, akkor találkozik legelőször a munkamenetszintű `Session("counter")` változóval. Mivelhogy ez a változó ebben a pillanatban még

nem létezik, a szerver létrehozza és ellátja az alapértelmezett 0 értékkel. Ezután következik az említett kifejezés kiértékelése, melynek értéke most **True** lesz. Következésképpen a munkamenetszintű `Session("counter")` változó értéke megváltozik 1-re és megjelenik a szöveg, mely tudósítja a látogatót, hogy a lapot most töltötte le először. Ha most frissítjük a lapot, a fenti kódot a szerver másodszor hajtja végre. Most viszont a `Session("counter")` munkamenetszintű változó már létezik a szerver memóriájában és értéke 1 (vagy egy nagyobb szám, ha többszörösen frissítettük a lapot), tehát a `Session("counter") < 1` kifejezés értéke **False**. Miután a `Session("counter")` értékét eggyel megemeltük, kiadjuk az ismételt látogatásról szóló szöveget. Mindét esetben a `Session.SessionID` tulajdonság ugyanaz.

Kattintsunk most a Befejezni kapcsolatra. Ezzel a `Session.Abandon_vbs.asp` lapot kérjük a szervertől. A lap forráskódja itt látható:

```
<%
 Response.Write( _
 "Ön befejezte ezt a munkamenetet (SessionID=<b>" & _
 Session.SessionID & "</b>).<br>" )
 Session.Abandon()
%>

<a href="Session.SessionID_vbs.asp">Új munkamenet </a>
```

A lap arról értesíti a látogatót, hogy befejezte az aktuális munkamenetet, amit a szerver a `Session.Abandon` metódus meghívásával végre is hajt. Egyúttal az összes munkamenetszintű változók, így a `Session("counter")` változó is, ki lesznek ürítve a szerver memóriájából. Ha most az Új munkamenet kapcsolatra kattintunk, újból letöltjük a `Session.SessionID_vbs.asp` lapot. A `Session("counter")` változó ki lett ürítve a szerver memóriájából, és amikor a szerver először találkozik vele, hozzárendeli az alapértelmezett 0 értéket. Az új munkamenet-azonosítónak most egy új értéke van.

Mielőtt továbbmennénk, érdemes észrevenni, hogy egy kis trükköt igényel a `Session.SessionID_vbs.asp` ASP lap JScript megfelelője. A VBScript a változókat, így a `Session("counter")` változót is, alapértelmezett értékekkel látja el. A JScript esetében ez egészen másképpen van. Ebben az esetben, ameddig a `Session("counter")` változóhoz nem rendeltünk értéket, addig **null** értékkel tartja számon a szerver. Ennélfogva nem használhatjuk az alábbi konstrukciót:

```
if (Session("counter")<1){
//...
}else{
//...
}
```

Ehelyett ellenőriznünk kell, hogy `Session("counter")` értéke nem-e **null**, vagy használhatjuk a JScript `isNaN(`*kifejezés*`)` függvényét, melynek visszatérési értéke **true**, ha *kifejezés* nem egy szám és **false** különben.

2.5 A `Request` objektum

Mint tudjuk, egy ASP alkalmazás folyamán egy élénk párbeszéd alakul ki az ügyfélgép, azaz a böngésző és a szerver között. A böngésző kérésekkel fordul a szerverhez, a szerver pedig a kért lapot küldi válaszként a böngészőnek. A válasz, tehát a lekért lap tartalma HTML kód, ami vagy eredetileg része a lekért lapnak vagy pedig az ASP `Respose` objektum `Write` metódusával kerül a lapba. Ahhoz, hogy ASP alkalmazásokat fejleszteni tudjunk, természetesen tudnunk kell, mit kérdez a böngésző a szervertől. Ehhez a `Request` objektum nyújt segítséget.

A `Request` objektum segítségével megtudhatunk lényegében mindent, amit a HTTP kérés magában foglal. Így megtudhatjuk a böngészőt, illetve a felhasználót azonosító HTML változók értékét, a böngésző álalt tárolt sütik (*cookies*) tartalmát, a lekért lap URL-jéhez kapcsolt *név* = *érték* párokat vagy a HTML `<FORM>` elemében felhasznált vezérlők nevét és értékét. Ha a kért lapot egy biztonságos csatornán (*secure channel*) keresztül kapjuk, mit például a Secure Sockets Layer (SSL) csatorna, akkor a digitális bizonylat (*digital certificate*) adatait is rendelkezésünkre bocsátja a `Request` objektum.

A `Request` objektum csak egy metódussal (`BinaryRead`) és egy tulajdonsággal (`TotalBytes`) viszont, igazi szerepéhez mérten, öt gyűjteménnyel rendelkezik:

- `ServerVariables`
- `QueryString`
- `Form`
- `Cookies`
- `ClientCertificate`

Ebben a szakaszban a `Request` objektumot fogjuk megismerni.

2.5.1 A `Request` objektum `BinaryRead` metódusa és `TotalBytes` tulajdonsága

A `BinaryRead` metódus és a `TotalBytes` tulajdonság első tekintetre elég furcsának tűnnek. Ha egy HTML lap `<FORM>` eleme a `POST` metódussal küldi el kérését a szerverhez, akkor a `TotalBytes` tulajdonság visszaadja a kérés hosszát bájtan:

```
intTotalBytes = Request.TotalBytes ' a kérés hossza bájtban
```

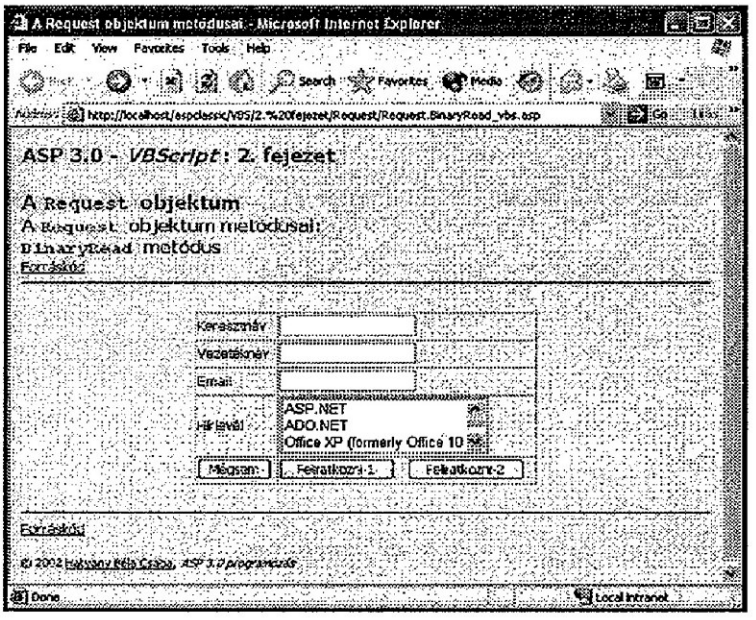
a `BinaryRead` metódus viszont, annyi bájtot olvas a kérésből, amennyit a metódus egyetlen egy argumentumában megszabunk:

```
Bstr = Resquest.BinaryRead(10)
```

A `BinaryRead` metódus argumentumának kisebbnek vagy egyenlőnek kell lenni a `TotalBytes` tulajdonság értékével. A `BinaryRead` metódus által visszaadott érték, amit a fenti példában a `Bstr` változóba tároltunk, nem egy karakterlánc. Típusa egy úgynevezett biztonságos tömb (*safe array*). Egy ilyen tömb struktúrájában hordozza a dimenziói számát és a dimenziók határát is. Nevezzük el a `Bstr` bájtokból álló biztonságos tömböt bájtsornak. Egy bájtsorral természetesen nem végezhetünk el olyan műveleteket, mint a karakterláncokkal és nem alkalmazhatunk karakterláncokra alkalmazható függvényeket sem. Mire jó akkor ez a metódus? – vetődik fel a jogos kérdés.

A `BinaryRead` metódust és a `TotalBytes` tulajdonságot leggyakrabban akkor alkalmazzuk, amikor egy állományt a HTTP protokoll segítségével fel akarjuk tölteni a szerverre. Ezzel a kérdéssel egy későbbi fejezetben foglalkozunk, ahol egy idevágó példát is látni fogunk. Most nézzünk meg a metódus és a tulajdonság egy egyszerű alkalmazását, ami megvilágítja, miként működik a `BinaryRead`.

Keressük meg `Request.BinaryRead_vbs.asp` állományt a jelen szakasz megfelelő mappájában és töltsük be a böngészőnkbe! Az eredményt a 2.17. ábrán láthatjuk.



2.17. ábra A `Request.BinaryRead_vbs.asp` *lap a böngészőben*

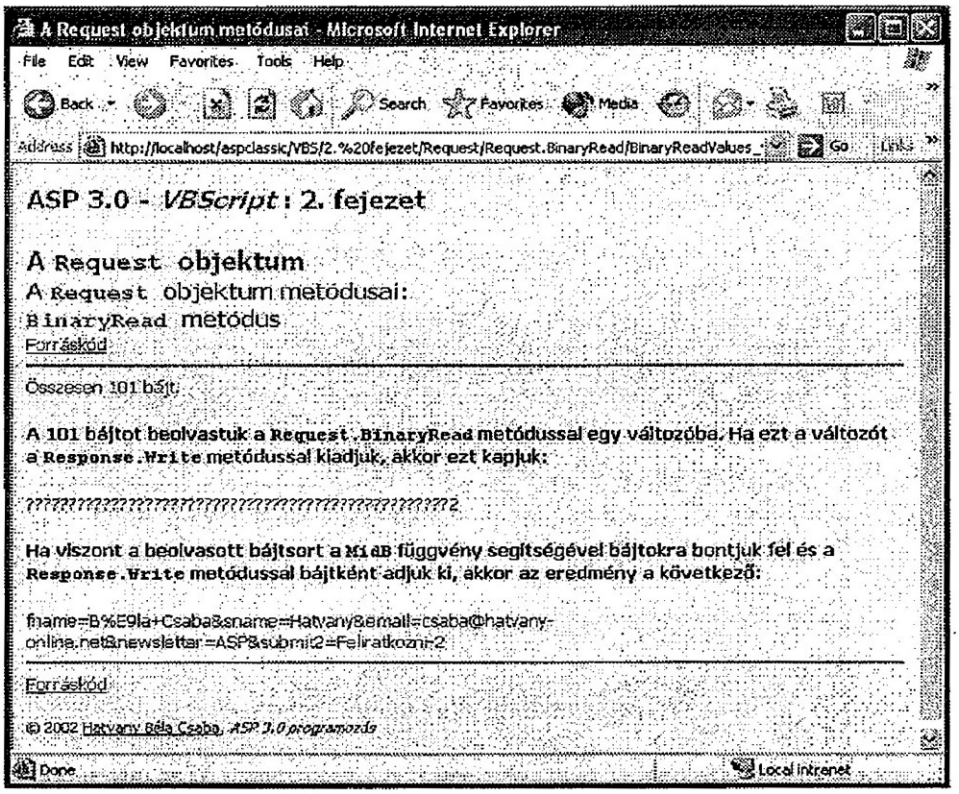
Mint látjuk, egy kérdőívről van szó. Töltsük ki a kérdőívet az ábrán látható adatokkal (Keresztnév: Béla Csaba, Vezetéknév: Hatvany, Email: csaba@hatvany-online.net, Hírlevél: ASP.NET)! Ha a **Feliratkozni-2** parancsgombra kattintunk (egyelőre ne tegyük ezt, várjunk még egy keveset, ameddig szólok), akkor egy kérést küldünk a szer-

verhez. A kérést a `Request.BinaryRead_vbs.asp` lapon elhelyezett `<FORM>` elem `action` és `method` attribútumaival határozzuk meg:

```
<FORM action=Request.BinaryRead/BinaryReadValues_vbs.asp
method=POST>
```

Amint látjuk a `Request.BinaryRead/BinaryReadValues_vbs.asp` lapot kérjük és a `POST` metódust használjuk. A `POST` metódus felküldi a szervere a `<FORM>` elemben tartalmazott HTML vezérlők nevét és értékét. Most kattintsunk a **Feliratkozni-2** parancsgombra és nézzük meg a szervertől leküldött lapot (2.18. ábra)!

Legelőször arról értesülünk, hogy összesen 101 bájtot küldünk a szerverre. Ezeket a bájtokat a `Request.BinaryRead` metódussal beolvassuk egy bájtsorba, majd *helytelenül* (de ez most egy kísérlet!) kiadjuk a (kizárólagosan karakterláncokra alkalmazható) `Response.Write` metódussal. Az eredmény nem mond sokat. Másodszor a bájtsort a `MidB` függvénnyel egyes bájtokra bontjuk fel és bájtként adjuk ki. Ebben az esetben az eredményből már jóval több érthető.



2.18. ábra A `BinaryRead` *metódus alkalmazása*

Nézzük meg közelebbről, mi történt itt. A második esetben felismerhetjük az előző lap `<FORM>` elemében szereplő HTML vezérlők nevét és értékét *`név = érték`* párokban:

```
fname=B%E91a+Csaba
sname=Hatvany
email=csaba@hatvany-online.net
```

```
newsletter=ASP
submit2=Feliratkozni-2
```

Megnézhetjük, hogy mi történik, ha a **Feliratkozni-1** gombra kattintunk?

A *név* = *érték* párok a & karakterrel vannak elválasztva. Ha egy *név*- vagy *érték*-ben egy szóköz volt, akkor itt egy + karakterrel van helyettesítve, ha pedig egy *név*- vagy *érték*-ben egy illegális karakter szerepelt, akkor itt az URL kódjával van helyettesítve (például é URL kódja %E9, ANSI decimál kódja 233 és ANSI hexadecimál kódja E9 = 14 × 16 + 9). Röviden a *név* = *érték* pársor & karakterrel elválasztva URL kódjával jelenik meg.

Számoljuk meg, hány karaktert tartalmaz a *név* = *érték* pársor, ügyelve arra, hogy az illegális karakterek helyett az megfelelő URL kódot vegyük. Az eredmény pontosan 101 = 2 × 50 + 1. Számoljuk meg hány karakter van az „értelmetlen" sorban. Az eredmény pontosan 51 = 50 + 1. Mivel magyarázható ez az összefüggés?

Tudnunk kell, hogy ASP belsőleg az egyes karaktereket két bájttal, az úgynevezett Unicode szerint reprezentálja. A szerverhez felküldött kérés viszont a karaktereket az ANSI kódolás szerint egy bájttal reprezentálja. Mit láttuk 101 karakterből álló kérést küldtünk fel a szerverre és ezek ANSI kódja érkezett meg a 101 bájtban. A 101 bájból álló bájtsort a Bstr változóba tároltuk.

Most hibásan, de tudatosan, visszaküldtük ezt a bájtsor a Response.Write metódussal a böngészőhöz. A Response.Write metódus viszont egy közönséges karakterláncot vár, ahol minden egyes karakter a Unicode szerint két bájból áll. Ennek megfelelően a kapott bájtsort két bájtként olvassa és értelmezi, tehát kiad 100/2 + 1 karaktert. Igen ám, de a kettesével olvasott bájtoknak (két különböző karakter ANSI kódja) legtöbbször nincs Unicode értelmük és akkor Response.Write egy ? karaktert jelenít meg. Ez a magyarázata az első 50 kérdőjelnek. Az utolsó bájthoz Response.Write még hozzátesz egy üres bájtot, hogy kettőt beolvasson. Esetünkben az utolsó bájt a 2 karakter ANSI kódja. Ha hozzáteszünk egy üres bájtot, akkor ez a két bájt magadja a 2 karakter Unicode kódját. (Az első 128 karakter Unicode kódja egyenlő a megfelelő karakter ANSI kódjával plusz egy üres bájt.) Ez a magyarázata a kérőjelek után megjelenített 2 karakternek. Néha megeshet, hogy két egymásután következő bájt a bájtsorból véletlenül egy Unicode kóddal egybeesik. Ilyen esetben megjelenik egy karakter, de aligha a megfelelő.

Ha viszont a bájtsort bájtként felbontjuk a MdiB függvénnyel, akkor MidB a beolvasott ANSI kódnak megfelelő Unicode kódját adja vissza és ezt továbbítjuk a Response.Write metódusnak megjelenítésre. Ez a magyarázata a kiolvasható karaktersornak.

A `Request.BinaryRead_vbs.asp` példának nincs JScript megfelelője.

A `TotalBytes` tulajdonság és a `BinaryRead` metódus alkalmazásával a 6. fejezetben fogunk foglalkozni. Azonban már most megállapíthatjuk, hogy jó szolgálatot tesznek az állományok a szerverre való felküldése esetében. Ha egy HTML `<FORM>` elembe beépítünk egy `<INPUT name=alomany type=file>` elemet, akkor a szerverre küldött bájtsorban a *név = érték* párok között szerepel egy pár, amelyik a vezérlő nevét tartalmazza és értéke a vezérlőbe írt állomány elérési útjából és az állomány egész tartalmából áll. A `TotalBytes` tulajdonság és a `BinaryRead` metódus segítségével kiszűrhetjük az állomány tartalmát és a szerverre írhatjuk.

Erre a kérdésre később fogunk visszatérni. Most nézzük meg a `Request` objektum gyűjteményeit.

2.5.2 A `Request` objektum gyűjteményei

Amikor a kliens oldalról egy kéréssel fordulunk a szerverhez, akkor a tényleges kérés mellett egy sor előre megadott értéket küldünk tovább. Ezek az értékek az úgynevezett szerverváltozókban vannak tárolva. A `Request` objektum `ServerVariable` gyűjteménye tartalmazza a szerverváltozókat és bármikor lekérdezhetők:

```
Request.ServerVariables(szerverváltozó_név)
```

ahol *szerverváltozó_név* egy karakterlánc, melynek értéke a szerverváltozó neve.

Az összes szerverváltozó neve és magyarázata a Függelékben található. Ezekből csak egy párat említek:

- `LOGON_USER` — a felhasználó bejelentkezési neve a Windows rendszerbe
- `HTTP_USER_AGENT` — egy karakterlánc, mely leírja a kérést intéző böngésző és operációs rendszer típusát
- `HTTP_ACCEPT_LANGUAGE` — a kliens által támogatott természetes nyelv
- `PATH_INFO` — az aktuális ASP lap virtuális elérési útja
- `PATH_TRANSLATED` — az aktuális ASP lap fizikális elérési útja
- `QUERY_STRING` — a teljes lekérdezés, ami a jelenlegi lap URL címéhez `?`-lel van kötve
- `REQUEST_METHOD` — a kérés metódusa: `GET`, `HEAD` vagy `POST`

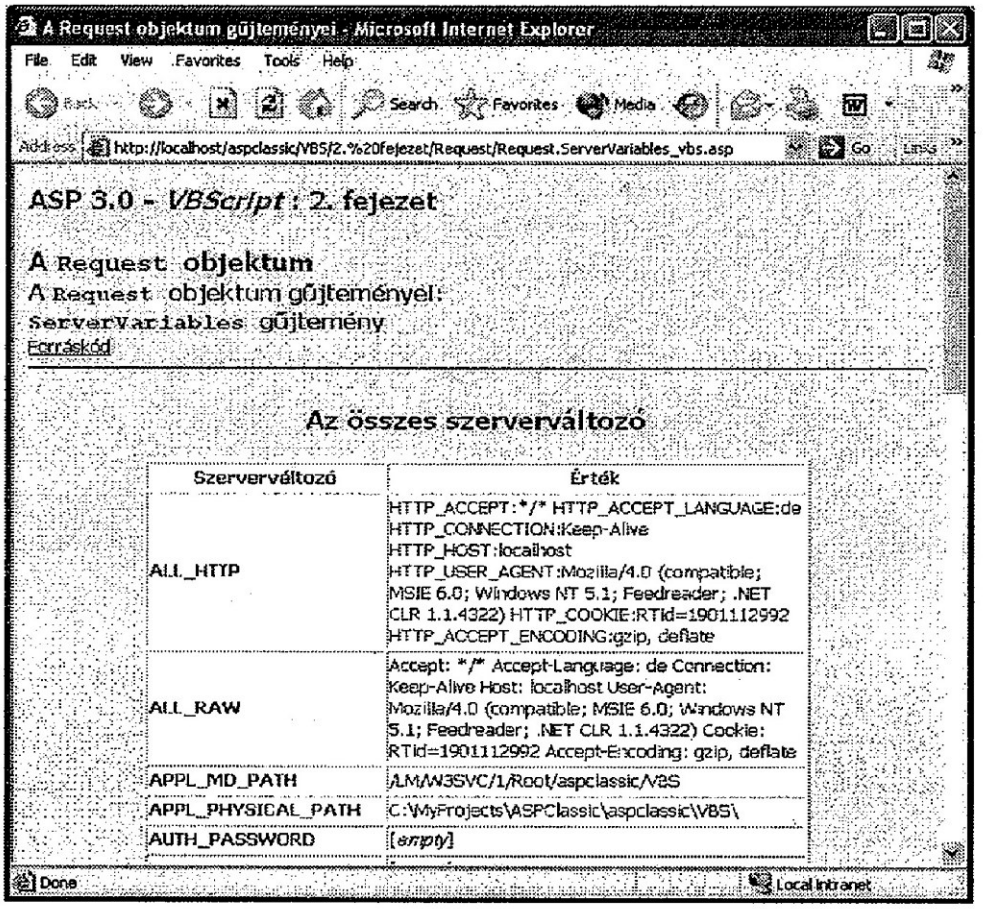
Ebből a rövid felsorolásból is kitűnik, milyen fontos tudnunk egyes szerverváltozók értékét. A `HTTP_USER_AGENT` értéke alapján a lapot dinamikusan különböző böngé-

sző típusokhoz igazíthatjuk, a `HTTP_ACCEPT_LANGUAGE` értéke maghatározhatja, hogy lapunk szövege magyarul vagy például angolul jelenjen meg.

A szerverváltozóban tárolt értékeket fejlécelemnek (*header*) is szokták nevezni. A következő szakaszban látni fogjuk, hogy a `Response.AddHeader` metódussal, az előre definiált fejlécelemeken túl, egyéni, úgynevezett felhasználói fejlécelemeket is definiálhatunk. A `ServerVariables` gyűjtemény segítségével a felhasználói fejlécelemeket is lekérdezhetjük. Ha a `Response.AddHeader` metódussal például a `PreferedSize` nevű és `"middle"` értékű fejlécelemet definiáljuk, akkor ennek a fejlécelemnek az értékét a következőképpen kérdezhetjük le. A fejlécnév elé a `"HTTP_"` karakterláncot iktatjuk, a fejlécnévben pedig minden egyes - (kötőjel) karaktert _ karakterre változtatjuk:

```
pref = Reqest.ServerVariables("HTTP_PREFERED_SIZE")
```

A `ServerVariables` gyűjtemény, a gyűjteményeknél megszokott `Count`, `Item` és `Key` tulajdonságokkal rendelkezik, ahol `Item` az alapértelmezett tulajdonság.



2.19. ábra A szerverváltozók

Különös szerepet játszanak az `ALL_HTTP` és `ALL_RAW` szerverváltozók. Mindkettő egy karakterláncot tartalmaz, mely az összes (alapértelmezett és felhasználói) fejlécelem-

ből áll *név* = *érték* alakban. A kettő között a különbség abban áll, hogy míg ALL_RAW a fejlécelem eredeti nevét használja, addig ALL_HTTP a HTTP neveket használja, azaz a fent leírt név-transzformációt végrehajtja.

A Request.ServerVariables_vbs.asp lap felsorolja az összes szerverváltozót (2.19. ábra).

A legtöbb ASP alkalmazás egy HTML <FORM> elemben HTML vezérlők segítségével adatokat kér a felhasználótól. A felhasználó kitölti az általa látott űrlapot és megnyom egy parancsgombot vagy egy képet és ezzel felküldi a beadott adatokat a szerverre. Itt egy másik ASP lap feldolgozza az adatokat és elkészíti a válaszlapot.

Az következő táblázat összefoglalja a HTML <FORM> elembe elhelyezhető HTML vezérlőket és az általuk felküldött adatokat:

Vezérlő	A továbbküldött adat
<INPUT type="checkbox"> <INPUT type="radio">	Ha ki van jelölve/választva, akkor a value attribútum értéke, vagy "on", ha a value attribútum nincs megadva. Ha nincs kijelölve/kiválasztva, nem küld értéket tovább.
<INPUT type="hidden">	A value attribútum értéke, amennyiben meg van adva.
<INPUT type="image"> <INPUT type="submit">	Ha meg lett kattintva, a value attribútum értéke, amennyiben meg van adva.
<SELECT>	A kiválasztott OPTION elem value attribútumának értéke, vagy az elem szövege, ha a value attribútum nincs megadva.
<SELECT MULTIPLE>	A kiválasztott OPTION elemek value attribútumának értékei vesszővel elkülönítve, vagy ezen elemek szövege vesszővel elkülönítve, ha a value attribútum nincs megadva.

Ha azonos típusú és azonos nevű elemek értékeket küldenek a szerverhez, akkor a közös nevet a megfelelő értékek követik vesszővel elkülönítve.

A fenti HTML elemeken kívül az <INPUT type="file"> vezérlő is küldhet adatokat a szerverhez. Sajátos mivolta miatt ezzel a vezérlővel egy későbbi, a 6. fejezetben foglalkozunk.

A HTML <FORM> elem action attribútumával megadjuk a felküldött adatokat feldolgozó ASP lap virtuális elérési útját, a method attribútummal pedig a felküldés módját. A method attribútum két értéket vár: GET és POST. Ha a method attribútumot elhagyjuk, akkor az adatok a GET metódussal kerülnek a szerverre. Mindkét metódus

esetében a felküldött adatok *név = érték* párokként lesznek a szerverre küldve, ahol *név* a vezérlő neve és *érték* a vezérlő által továbbított adat. A GET és POST metódus közötti különbség, többek között abban áll, hogy a felküldött adatok a szerveren a Request objektum különböző gyűjteményeibe kerülnek. Ha a GET metódust használjuk, akkor az adatok a QueryString gyűjteménybe, ha viszont a POST metódust alkalmazzuk, akkor az adatok a Form gyűjteménybe kerülnek.

Egy másik fontos különbség a két metódus között a lekérdezési sorozat vagy lekérdezési lánc. A POST metódus esetében ez egy üres karakterlánc. A GET metódus alkalmazásakor viszont az ASP a lekért lap URL-jéhez a ? karakterrel egy *név = érték* pársorozatot kapcsol, amelyben a *név = érték* párokat a & karakterrel különíti el egymástól. Ezt a karakterláncot nevezzük lekérdezési sorozatnak vagy lekérdezési láncnak.

Térjünk most át a Request metódus gyűjteményeinek megismerésére!

A Request.QuerySring gyűjtemény alkalmazásának szintaxisa

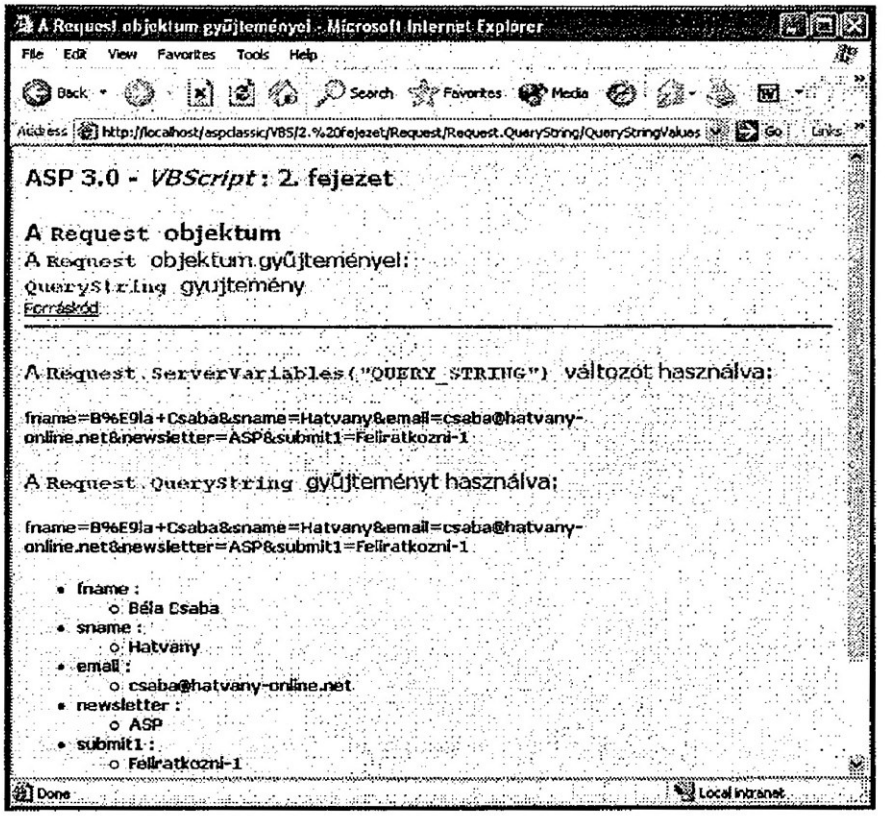
```
val = Request.QuerySring([név])
```

ahol *név* egy HTML vezérlő neve egy opcionális argumentum. Ha az argumentumot nem adjuk meg, akkor a metódus az egész lekérdezési láncot adja vissza. Ha az argumentum a lekérdezési láncban szereplő vezérlőnevet jelöl, akkor a visszatérési érték a szóban forgó vezérlő értéke a lekérdezési láncban, ha viszont az argumentum egy név, mely nem szerepel a lekérdezési láncban, akkor a metódus az üres karakterláncot adja vissza.

A Request.QueryString1_vbs.asp lap egy egyszerű példa a QuerySring gyűjtemény alkalmazására. Miután kitöltöttük az űrlapot, kattintsunk például a **Feliratkozni-1** gombra. Az eredményt a 2.20. ábrán láthatjuk.

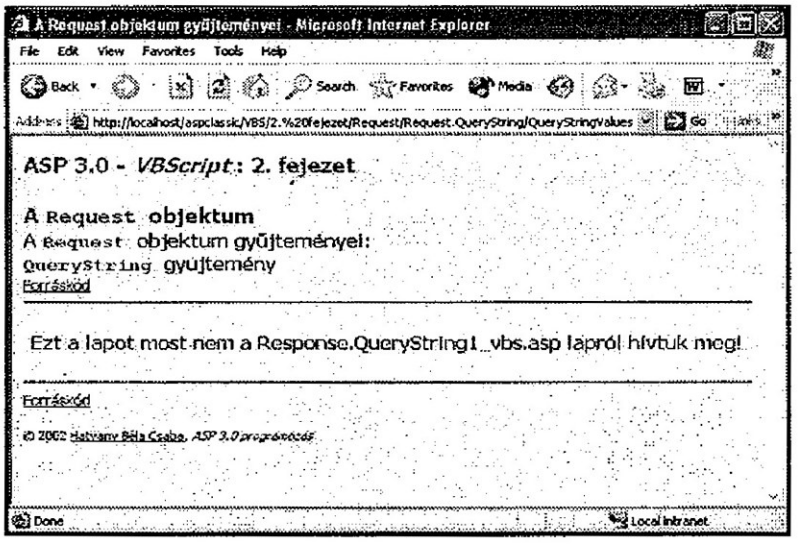
Érdemes észrevenni, hogy a QuerySring gyűjtemény alkalmazása argumentum nélkül ugyanazt az eredményt adja, mint a ServerVariables gyűjtemény QUERY_STRING eleme.

Fontos tudni, hogy a GET metódussal lekért lap URL-jéhez csatolt lekérdezési lánc tisztán kiolvasható a lekérdezett lap megjelenése után az **Internet Explorer** címmezőjében. Bizalmas adatokat ezért sose továbbítsunk a GET metódussal.



2.20. ábra A `QueryString` *gyűjtemény*

Ha a 2.20. ábrán látható lap Forráskód hivatkozására, majd a forráskódot megjelenítő Megjelenítés hivatkozására kattintunk, akkor ismét lehívjuk a 2.20. ábrán látható lapot, de ezúttal nem küldünk semmilyen lekérdezési láncot a szerverre. Ezt az esetet a `QueryString.Count` tulajdonság megvizsgálásával a lap kódjában felhasználhatjuk a megjelenítés megfelelő kialakítására, amint azt a 2.21. ábrán láthatjuk.



2.21. ábra A `Request.QueryString.Count` *tulajdonság megadja a továbbított adatos számát*

A `QueryString` gyűjtemény a gyűjteményekre jellemző `Count`, `Item` és `Key` tulajdonságokkal rendelkezik, ahol `Item` az alapértelmezett tulajdonság.

Lekérdezési láncokat közvetlenül, azaz a HTML <FORM> elem nélkül is továbbíthatunk a szervernek. Ebben az esetben a HTML <A> elem `href` attribútumába magunknak kell csatolni a lekérdezett lap URL-jéhez a továbbítandó lekérdezési láncot. A lekérdezett lapban alkalmazhatjuk a `QueryString` gyűjteményt, hogy eldöntsük, miként járunk el tovább.

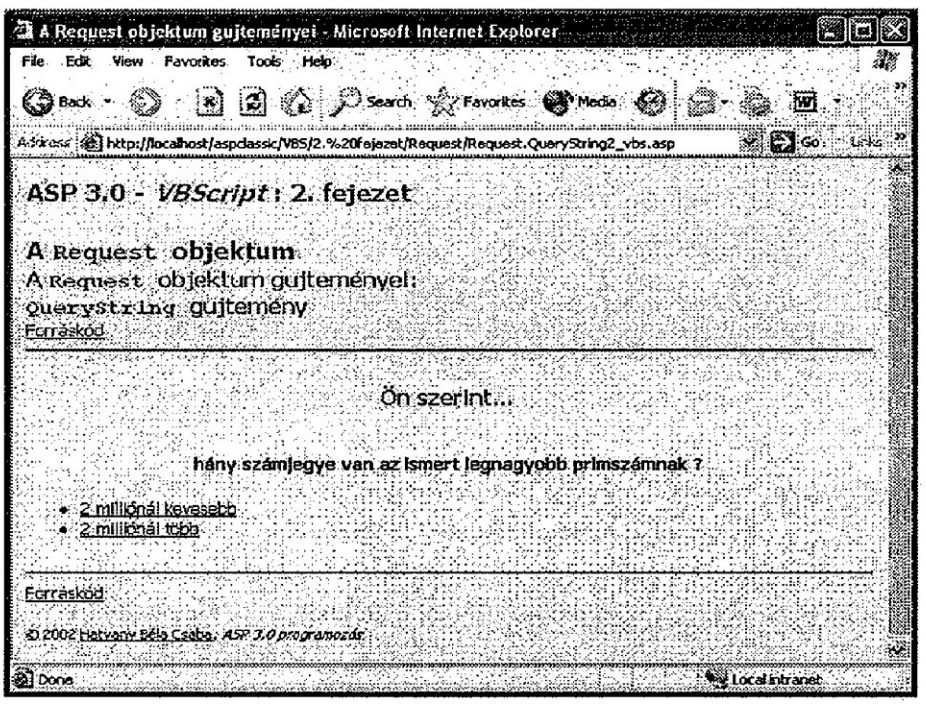
Íme egy egyszerű példa:

```
Kattintson a helyes válaszra! <BR>

Ön szerint Ausztrália fővárosa:
<A href="valasz.asp?valasz=helyes">Canberra </A><BR>
<A href="valasz.asp?valasz=hamis">Melbourne </A><BR>
<A href="valasz.asp?valasz=hamis">Sidney </A><BR>
```

A `valasz.asp` lapon eldöntjük, hogy mit írunk ki válaszként

```
Az Ön válassza
<%
 Response.Write = Request.QueryString("valasz")

%>
!
```

A `Request.QueryString2_vbs.asp` lap egy valamivel bővebb, de ugyanilyen egyszerű példát mutat be (2.22. ábra).



2.22. ábra Lekérdezési lánc az <A> elemben

Ebben az esetben a lekérdezési lánc még a lekérdezett lap megjelenítése előtt is látható. Elegendő, ha az egyik hivatkozás fölé tesszük az egeret. Ezért ez a megoldás

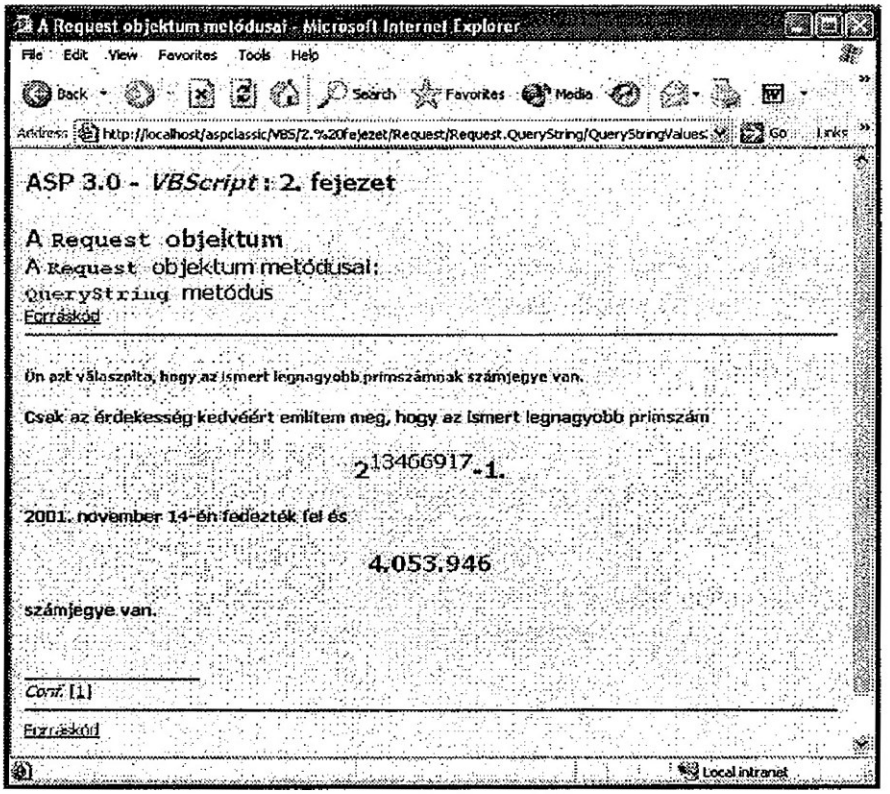
sem használható bizalmas adatok esetében. Ez a kis alkalmazás viszont kitűnően alkalmas arra, hogy megvizsgáljuk, mire kell vigyázzunk, ha adatokat a lekérdezési lánccal továbbítunk.

Töltsük le még egyszer a `Request.QueryString2_vbs.asp` lapot! Ahelyett, hogy egy hivatkozásra kattintanánk, adjuk be a címmezőbe a következő címet:

```
http://localhost/aspclassic/VBS/2.%20fejezet/Request/
Request.QueryString/QueryStringValues2_vbs.asp
```

(Ha túl nehéz beadni ezt a hosszú címet, használjuk a `TestQueryString-a.asp` lapot ahova beépítettem ennek a lapnak a lehívását.)

Így elérjük azt, hogy a válaszlapot lekérdezési lánc nélkül töltsük le. Az eredmény ennek megfelelően értelmetlen, vagy legalábbis furcsa (2.23. ábra).



2.23. ábra Ha nem ellenőrizzük a `QueryString` *gyűjteménybe küldött adatokat, ilyen meglepetésben is részünk lehet*

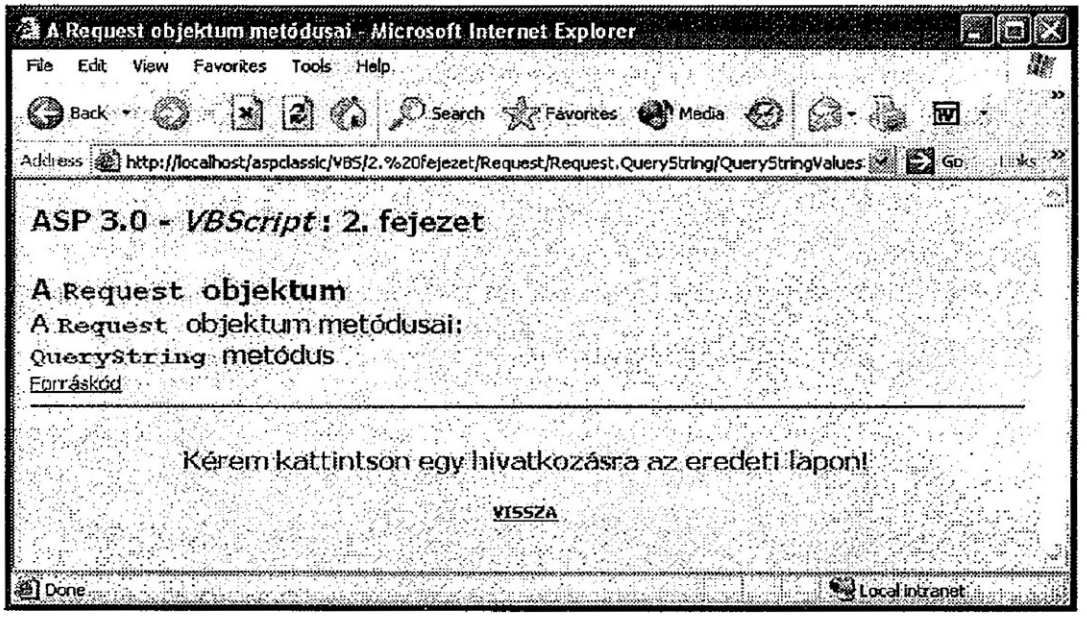
Az példából adódó tanulság, hogy mindig ellenőrizzük a szerverre küldött adatokat, legfőképpen akkor, ha azok a felhasználótól jönnek. Esetünkben a `Request.QueryString/QueryStringValues2_vbs.asp` kódját csupán egy pár sorral kell kiegészítenünk, hogy elkerüljük a 2.23. ábrán látható hiányos szöveget:

```
If Request.QueryString("ans")="" Or _
Request.QueryString("val")="" Then
        Response.Write("<CENTER><H2><FONT " & _
```

```
"color=red>Kérem válasszon ki egy esetet " & _
"az eredeti lapon!</FONT></H3>")" & _
        Response.Write("<A " & _
"href=""../Request.SueryString2_vbs.asp"">" & _
"VISSZA</A></CENTER>")
        Response.End
End If
```

Az itt használt `Response.End` metódusról később még szó esik. Most csak annyit jegyezzünk meg, hogy a metódus meghívása után ASP nem olvassa tovább a lap tartalmát és megszünteti a további forráskód megjelenítését, illetve értelmezését.

A `Request.QueryString/QueryStringValues3_vbs.asp` állomány tartalmazza az így módosított forráskódot. Ez a lapot hívja le a `Request.QueryString3_vbs.asp` lap. Ha az **Internet Explorer** címmezőjébe az előbbi címet beadjuk (vagy a `TestQueryString-b.asp` lapon levő hivatkozást használjuk), akkor előbbi értelmetlen szöveg helyett egy figyelmeztetés jelenik meg (2.24. ábra).



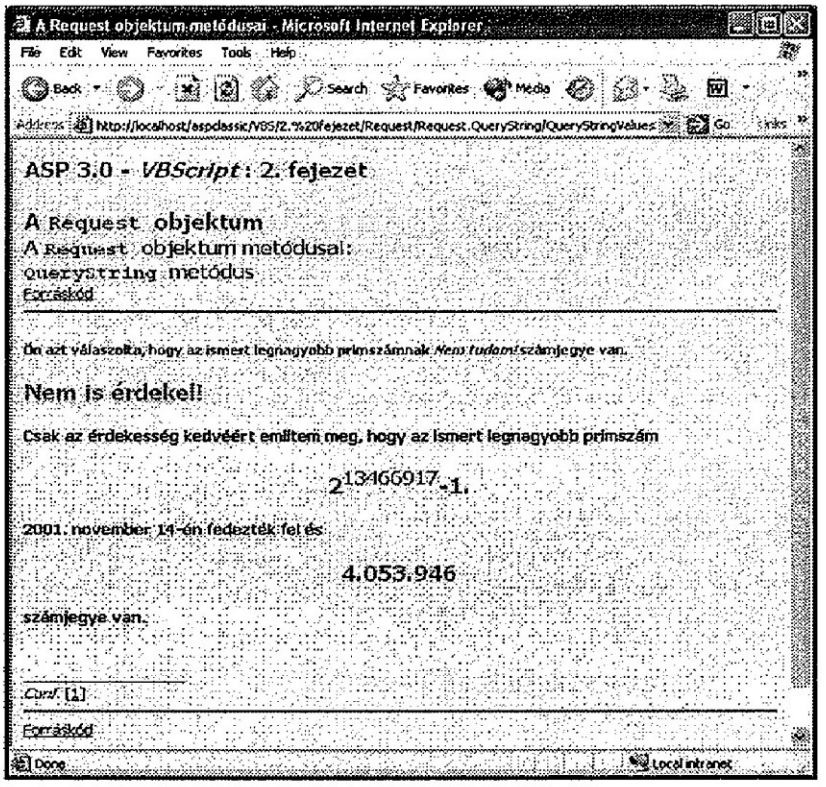
2.24. ábra Ha a `QueryString` *adatai nem felelnek meg elképzelésünknek, kellőképpen reagálhatunk*

A `QueryString` gyűjteménybe felvett adatok ellenőrzésével, még nem tettünk meg mindent az alkalmazásuk megvédésére külső „támadásoktól". Az előbbi példában csak azt ellenőriztük, hogy a `QueryString` gyűjteménybe küldött két név (`ans` és `val`) értéke az üres karakterlánc vagy nem. Megtörténhet, hogy a lapunkat, a látható lekérdezési láncból kiindulva, egy egészen más értékekkel ellátott lekérdezési lánccal hívják le, mint ahogy azt mi terveztük. A már bizonyos fokig megvédett lapun-

kat (`Request.QueryString/QueryStringValues3_vbs.asp`) töltsük le, ezúttal a következő címmel:

```
http://localhost/aspclassic/VBS/2.%20fejezet/Request/
Request.QueryString/QueryStringValues3_vbs.asp?ans=
<font%20color=red><i>Nem+tudom!</i></font>&val=<H1>
<font%20color=red>Nem+is+%E9rdekel!
```

(vagy használjuk a `TestQueryString-c.asp` segédlapot). Az eredményt a 2.25. ábrán láthatjuk.



2.25. ábra A lapunkat egy újabb „támadás" érte

Példánkban a továbbküldendő `ans` és `val` nevekkel párosított értékek, a lap kódjában vannak rögzítve. A szóban forgó értékek a következő karakterláncok: az `ans` név esetében `"2 milliónál kevesebb"` vagy `"2 milliónál több"`, a `val` név esetében `"Sajnos, az Ön válasza nem helyes."` vagy `"Gratulálok, az Ön válasza helyes!"`. A „támadás" ellen úgy is védekezhetnénk, hogy a `QueryStringValues3_vbs.asp` lapon ezeket az értékeket ellenőrizzük. Ez a módszer azonban nem mindig alkalmazható, mivelhogy nem minden esetben ismerjük előre a lekérdezési láncba küldött értékeket. Itt egy másik „védekezési" lehetőséget alkalmaztam. A `HTTP_REFERER` szerverváltozó visszaadja a lehívó lap URL-jet, így megtudhatjuk, honnan hívta le a felhasználó a válaszlapot.

Egészítsük ki a Request.QueryString/QueryStringValues3_vbs.asp lap forráskódját a következő kódrészlettel:

```
Dim strHost, strReferer, strRefererHost, strRefererPage
strHost = Request.ServerVariables("HTTP_HOST")
strReferer = Request.ServerVariables("HTTP_REFERER")
 If strReferer<>"" Then
    strReferer = Right(strReferer, Len(strReferer) - _
          (InStr(1, strReferer, "://") + 2))
    strRefererHost = Left(strReferer, _
          InStr(1, strReferer, "/") - 1)
    strRefererPage = Right(strReferer,Len(strReferer) - _
          InStrRev(strReferer, "/"))
 End If
 If strRefererHost <> strHost Or _
   strRefererPage <> "QueryStringValues4_vbs.asp" _
 Then
    Response.Write("<CENTER><H2><FONT color=red>" & _
       "Kérem kattintson egy hivatkozásra az " & _
       "eredeti lapon!</FONT></H3>")
    Response.Write("<A href=" & _
       """../Request.QueryString4_vbs.asp""" & _
       ">VISSZA</A></CENTER>")
   Response.End
End If
```

A Request.QueryString/QueryStringValues4_vbs.asp lap tartalmazza a módosított forráskódot. Az így módosított lap ilyennemű „támadásokat" ki tud védeni. Ellenőrzésre használjuk a TestQueryString-d.asp segédlapot.

A továbbiakban vizsgáljuk meg a POST metódust. A POST metódussal felküldött adatok a Form gyűjteménybe kerülnek és ezzel a gyűjteménnyel kérdezhetjük le. Ebben az esetben a lekérdezési lánc természetesen üres. A Form gyűjtemény a már megszokott három tulajdonsággal rendelkezik: Count, Item és Key, ahol ismét az Item tulajdonság az alapértelmezett.

A Form gyűjteményt a következő szintaxis szerint alkalmazzuk:

```
val = Request.Form([név])
```

ahol *név* egy opcionális argumentum és egy HTML vezérlő nevét adja meg. Ha az argumentumot nem adjuk meg, akkor a metódus az összes *név* = *érték* párt & karakterrel egymástól elválasztva adja vissza. Ha az argumentum egy olyan vezérlő nevét jelöli, mely egy értéket továbbít, akkor a visszatérési érték a szóban forgó vezérlő értéke, különben a visszatérési érték az üres karakterlánc.

A `Request.Form1_vbs.asp` lap bemutatja a `Form` gyűjtemény használatát, ugyanazt a kérőívet használva, mint a `Request.QueryString1_vbs.asp` lap. A `Request.Form2_vbs.asp` és a `HTMLControls_vbs.asp` lap megmutatja, miként ellenőrizhetjük a felhasználó által beadott adatokat.

A `Request` objektum gyűjteményébe tárolt adatok kiolvasása sokkal bonyolultabb művelet, mint egy egyszerű változó esetében és ennek megfelelően több időt vesz igénybe. Ezért, ha egy gyűjteménybe tárolt értéket többször akarunk felhasználni, esetleg megváltoztatni, tanácsos az értéket egy közönséges változóba tárolni, hogy elkerüljük a gyűjteményből való többszörös kiolvasást.

Ritkán előfordulhat az az eset, amikor ismerjük az érték nevét, ami egy gyűjteménybe tárolva van, de nem tudjuk, hogy melyik gyűjteményben fordul elő. Egy ilyen eset például akkor áll elő, ha két lapon elhelyezett kérdőív ugyanazt az ASP lapot hívja meg, az egyik a `POST` a másik a `GET` metódust használva. Az első esetben az ismert név a `Form` gyűjteményben, a második esetben a `QueryString` gyűjteményben fog szerepelni. Ilyenkor fontos tudni, hogy például a *név* nevű értéket a következő meghívással érjük el:

```
val = Request(név)
```

és hogy ASP a `Request` objektum gyűjteményeit a következő sorrendbe kutatja át: `QueryString`, `Form`, `Cookies`, `ClientCertificate` és legutoljára `ServerVariables`.

A hagyományos HTTP protokoll mellett a szerver és az ügyfélgép kapcsolata biztonságos csatornán is létrejöhet. A legelterjedtebb biztonságos csatorna az SSL (*Secure Sockets Layer*). Az SSL csatorna protokollja a magas biztonságot azonosítással, rejtjelezéssel és adatintegritással hozza létre. Az azonosítás biztosítja, hogy az adatok csak egy „engedélyezett" ügyfélgép és úgyszintén „engedélyezett" szerver között közlekedjenek. A rejtjelezés arról gondoskodik, hogy az adatokat csak az engedélyezett ügyfélgép és szerver tudja elolvasni. Végül az adatintegritás felelős az adatok változatlan formába való megérkezésére a címzetthez. Ha egy kliens SSL protokollal kíván egy szervertől adatokat kérni vagy egy szerverre adatokat küldeni, akkor a kapcsolatot a megszokott `http://` helyett, a biztonságos kapcsolatot elindító `https://` protokollkéréssel kell kezdeményeznie. Egy biztonságos kapcsolat létrejötte után, mindegyik fél le tudja kérdezni a másik fél digitális biztonsági bizonylatának (*digital security certificate*) adatait. A digitális biztonsági bizonylatot egy harmadik fél, a *Certificate Authority* (CA) bocsátja ki.

A biztonságos weblapok fejlesztése nagyon fontos témaköre a webalkalmazásoknak, könyvünk keretét azonban meghaladja. Itt csak annyit jegyezzünk meg, hogy a digitá-

lis biztonsági bizonylat adatainak lekérdezése a `Request` objektum `ClientCertificate` gyűjteményével történik. Egy jó bevezető a digitális biztonsági bizonylatról a http://www.verisign.com/repository/ cím alatt érhető el.

A `Request` objektum hátramaradt gyűjteménye a `Cookies` gyűjtemény. Mivelhogy a `Response` objektumnak is van egy `Cookies` gyűjteménye, könnyebb megérteni a `Cookies` gyűjtemény szerepét, ha egy külön szakaszban tárgyaljuk. Ez fogjuk tenni a 2.7. szakaszban.

Előbb azonban vizsgáljuk meg közelebbről a mostanáig is sokat használt `Response` objektumot.

2.6 A `Response` objektum

Aligha lehet egy ASP alkalmazást elképzelni, mely ne használná a `Response` objektumot. Mi is mostanáig sokszor használtuk a `Response` objektum `Write` metódusát. Bizonyos értelemben a `Response` objektum a `Request` objektum ellentéte: arra használjuk, hogy a szervertől a kliensgépre küldjünk információt, illetve hogy a kliensgépre küldött információt, az úgynevezett HTML válasz elemeit elérhessük és ellenőrizhessük. Ennek megfelelően a `Response` objektum aránylag sok metódussal és tulajdonsággal és csak egy gyűjteménnyel rendelkezik.

A `Response` objektum metódusai a következők:

- `AddHeader`
- `AppendToLog`
- `BinaryWrite`
- `Clear`
- `End`
- `Flush`
- `PICS`
- `Redirect`
- `Write`

Tulajdonságai:

- `Buffer`
- `CashControl`
- `Charset`
- `ContentType`
- `Expires`
- `ExpiresAbsolute`
- `IsClientConnected`
- `Status`

Az egyetlen gyűjteménye a `Cookies` gyűjtemény.

2.6.1 A Response objektum metódusai

A `Write` metódussal már találkoztunk, így most rátérhetünk a többi metódus tárgyalására.

Az első fejezetben láttuk, miként lesz egy ASP lap a böngészőben megjelenítve. ASP értelmezi és feldolgozza az ASP lap tartalmát. A lapban talált HTML kódot kiegészíti az ASP szkript részeinek értelmezésével, melyeket beleiktat a HTML kódba a megfelelő helyre. Így létrejön egy teljesen és kizárólagosan HTML kódból álló adatsor, amit a szerver, pontosabban az IIS, visszaküld a böngészőnek megjelenítésre. Nézzük meg közelebbről, hogy keletkezik egy vegyesen HTML és ASP kódból álló adatsorból egy kizárólagosan HTML kódot tartalmazó adatsor?

Vegyünk egy egyszerű példát:

```
<HTML>
<BODY>
<H3>
A webszerveren a pontos idő
<% Response.Write " <FONT color=red>" & Time & _
                  "</FONT>" %>.
</H3>
</BODY>
</HTML>
```

Ez volt lényegében legelső példánk. Akkor röviden láttuk, hogy ebből a „vegyes" HTML és ASP kódból a böngészőhöz egy „vegytiszta" HTML kód jut el. Most vizsgáljuk meg, hogy képes ASP azt a HTML kódot létrehozni?

ASP azzal kezdi, hogy legelőször létrehoz egy üres HTML adatsort vagy adatáramlatot (*HTML data stream*). Ezt az egyelőre üres adatsort egy átmeneti tárolóba vagy pufferbe (*buffer*) helyezi el, az IIS pufferbe. A pufferbe mindig csak az adatsor végéhez lehet újabb adatokat beírni, a beírt adatok pontosan a beírásuk sorrendjében vannak és ezt a sorrendet nem lehet megváltoztatni. Éppen ezért, ASP legelőször a HTTP fejlécet (*HTTP header*) írja a pufferbe. Ha egy lap egy kérést intéz a szerverhez, akkor a szerver a válaszát a HTTP fejléccel kezdi. A fejléc egy karaktersor, mely bizonyos információt tartalmaz, amire a böngészőnek szüksége van. Általában az ASP alkalmazások programozásánál nem kell foglalkoznunk a fejléc tartalmával. (A `Response\ShowHTTPHeader_vbs.htm` lap egy VBScript szkriptet tartalmaz, mely megjeleníti a HTTP fejlécet.)

Miután ASP beírta a pufferbe a HTTP fejlécet, megvizsgálja a lap tartalmát. Ha HTML kódot talál, beírja a pufferbe. Példánk esetében, ASP elolvassa a lapunkat egé-

szen az először előforduló <% karakterpárig. Idáig csak HTML kódot talált, tehát beírja a pufferbe. Ebben a pillanatban a puffer (jelképesen) így néz ki:

```
<HTTP fejléc><HTML><BODY><H3>A webszerveren a pontos idő
```

Az első <% karakterpár rábukkanásakor ASP megkeresi a megfelelő %> karakterpárt és elkezdi a köztük talált ASP kód értelmezését. A mi esetünkben a `Response.Write` metódust találja, ami azt jelenti, hogy végeredményben egy karakterláncot kell a böngészőhöz küldenie. A karakterlánc három részből áll. A második rész egy függvény (`Time`) visszatérési értéke (tegyük fel, hogy `"19:27"`). ASP összeállítja a karakterláncot és beírja a pufferbe, mely ebben a pillanatban a következő HTML kódot tartalmazza:

```
<HTTP fejléc><HTML><BODY><H3>A webszerveren a pontos idő <FONT
color=red> 19:27</FONT>
```

A lap hátramaradt részében ASP már csak HTML kódot talál, tehát beírja a pufferbe:

```
<HTTP fejléc><HTML><BODY><H3>A webszerveren a pontos idő
<FONT color=red> 19:27</FONT>.</H3></BODY></HTML>
```

Most, hogy a lap értelmezése befejeződött, IIS kiürítheti a puffert és elküldheti a benne levő adatsort megjelenítésre a böngészőnek.

Hogy ez valóban így történik, arról a következő egyszerű, de nagyon furcsán kódolt, lap helyes megjelenítése győzhet meg:

```
<HT<%a="teszt"%>ML><BO<%b="lap"%>DY><H<%c= a &
b%>1><%Response.Write c%></H1></BODY></HTML>
```

Az **Internet Explorer**-ből ellenőrizhetjük, hogy a böngészőhöz küldött kód ez:

```
<HTML><BODY><H1>tesztlap</H1></BODY></HTML>
```

Ha a lapban több ASP kódrészlet van, a lap értelmezése és a HTML adatsor előállítása hasonlóképpen történik. Ha a puffer megtelik mielőtt a lap értelmezése véget ért volna, akkor a benne foglalt csonka adatsort az IIS elküldi a böngészőknek és kiüríti a puffert. ASP ott folytatja a puffer feltöltését, ahol abbahagyta. Tapasztalatból (azaz saját, sokszor hiányos kódunk helyes megjelenítéséből) tudjuk, hogy a böngészők elég „intelligensek" ahhoz, hogy ilyen csonka kódrészletekből csak annyit jelenítsenek meg, aminek HTML értelme van.

Előfordulhat azonban, hogy egyes ASP kódrészletek kiértékelése több időt vesz igénybe, tehát a legutolsó megjelenítéstől számítva, amikor a puffer legutoljára lett kiürítve, túl sok időt kell várnia a böngésző kezelőjének, ameddig egy újabb megjelenítést észlel a böngészőn. Ez főleg komplikált számítások vagy adatbázis lekérdezések

esetében fordul elő. Sokat segítene ilyen esetekben, ha a programozó irányítani tudná a puffer kiürítését, más szóval, ha rendelkezne olyan eszközökkel, melyekkel tetszése szerint egy nem teljesen feltöltött puffert ki tudna üríteni, esetleg egy már a pufferbe került kódsort kiüríthetne a pufferből anélkül, hogy elküldené a böngészőhöz vagy akár elérhetné, hogy a puffer már ne fogadjon be semmit.

Ilyen eszközöket bocsátanak rendelkezésünkre a `Response` objektum `Buffer` tulajdonsága és a `Flush`, `Clear` és `End` metódusai.

A `Buffer` tulajdonság **`True`** vagy **`False`** értékeket vehet fel és megszabja, illetve visszaadja a puffer működésének a módját.

Ha `Response.Buffer` értéke **`False`**, akkor a szerver amint kiértékelt egy kódrészletet, máris elküldi a böngészőnek megjelenítésre. Ilyenkor az szoktuk mondani, hogy a pufferelés ki van kapcsolva. Ha a pufferelés be van kapcsolva, tehát ha `Response.Buffer` értéke **`True`**, akkor a szerver csak azután küld kódot a böngészőhöz, miután az egész lap ki lett értékelve vagy a `Response` objektum `Flush` vagy `End` metódusát meghívtuk.

A `Buffer` alapértelmezett értéke **`True`**. Ez az ASP 3.0 verziójának egy újítása az előbbi verziókkal szemben, ahol alapbeállításban a pufferelés ki volt kapcsolva. A `Buffer` tulajdonság értékét csak akkor változtathatjuk meg, ha a kódsor bármely része, a HTTP fejlécet is ideértve, még nem lett elküldve a böngészőhöz. Más szóval, a `Buffer` értékét csak üres puffer esetében változtathatjuk meg. Ezért legegyszerűbb, ha a `Response.Buffer = érték` utasítást közvetlenül a <%@ ... > direktíva után használjuk vagy, ha az `Option Explicit` utasítás van ezen a helyen, akkor közvetlenül utána.

Ez a lap hiba nélkül jelenik meg:

```
<%
 a="teszt"
 b="lap"
 c= a & b
 Response.Buffer=False
%>
<HTML><BODY><H1>
<%Response.Write c%>
</H1></BODY></HTML>
```

de a következő egy hibát idéz elő:

```
<%
 a="teszt"
 b="lap"
 c= a & b
%>
```

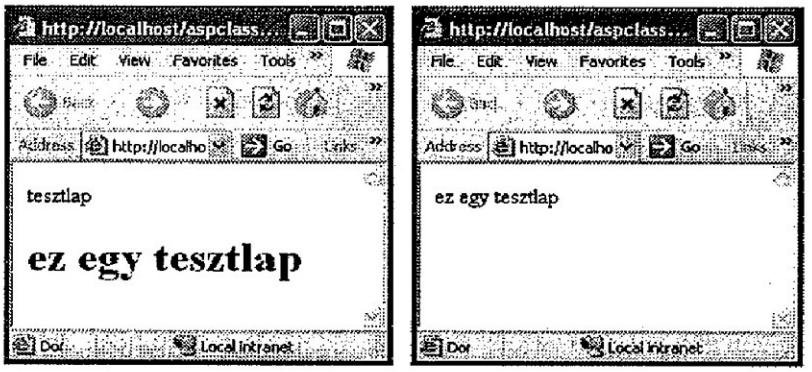
```
<HTML><%Response.Buffer=False%><BODY><H1>
<%Response.Write c%>
</H1></BODY></HTML>
```

Ha a pufferelés ki van kapcsolva, akkor a most bemutatásra kerülő `Flush`, `Clear` és `End` metódusok meghívása szintén egy hibát generál.

A `Flush`, `Clear` és `End` metódusok nem várnak paramétereket. A `Flush` metódus kiüríti a puffer tartalmát és elküldi a benne levő kódsort a böngészőhöz. Legtöbbször akkor használjuk, amikor laprészeket akarunk a böngészőhöz küldeni, például több időt igénybe vevő ASP kódrészlet előtt. A `Clear` metódus kiüríti a puffert, de nem küldi el a benne talált kódsort a böngészőhöz. Ha a puffer a HTTP fejlécet is tartalmazza (ez elsőként került be), akkor a `Clear` metódus a HTTP fejlécet nem üríti ki a pufferből. A pufferben levő kód nem kerül megjelenítésre, de az ASP változók megtartják értéküket és tovább használhatók.

Végezzünk el egy kísérletet! Jelenítsük meg a következő lapot:

```
<%
  Response.Buffer=True
  a="teszt"
  b="lap"
  c= a & b
%>
<HTML>
<%
Response.Write c
%>
<BODY><H1>
<%
'Response.Flush
'Response.Clear
Response.Write "ez egy " & c%>
</H1></BODY></HTML>
```



2.26. ábra A `Flush` *és* `Clear` *metódusok alkalmazása*

A lapot a 2.26. ábra bal oldalán láthatjuk. Most távolítsuk el a kommentárt jelző ' karaktert a `Response.Clear` hívás elől és frissítsük a lapot. Az eredményt az ábra jobb oldali képén láthatjuk. Közvetlenül a `Response.Clear` metódus meghívása előtt a puffer tartalma a következő:

```
<HTTP fejléc><HTML>tesztlap<BODY><H1>
```

A `Response.Clear` metódus kiüríti a puffert egészen a HTTP fejlécig és a puffer tartalma nem kerül a böngészőhöz: az első szöveg (tesztlap) nem jelenik meg. Rákövetkezően belekerül a pufferbe a hátramaradt kódrészlet:

```
<HTTP fejléc>ez egy tesztlap</H1></BODY></HTML>
```

Ez a kódsor kerül a böngészőhöz megjelenítésre. A böngésző „megbocsátja" nekünk, hogy a <HTML> és <BODY> elemek kezdőrészét nem kapta meg (akár az egész elem elmaradását is elnézi nekünk). A <H1> elem kezdőrészére szüksége van, mert tudnia kell, hol kezdődik a <H1>-es cím. Mivelhogy ezt nem találja meg, megjeleníti a pufferben talált szöveget az alapbeállítása szerint. Végül távolítsuk el a kommentárt jelző jelet a `Response.Flush` elől is. Ez az utasítás az elsőként tárgyalt puffertartalmat azonnal elküldi a böngészőhöz, így a `Response.Clear` üres puffert talál, amit „kiürít". Majd bekerül a pufferbe a maradék kódrészlet is, és a lap értelmezésének befejeztével ez is el lesz küldve a böngészőhöz. Egy újabb frissítés után a lap ismét úgy néz ki, mint az ábra bal oldali képén.

Az `End` metódus kiüríti a puffer tartalmát, elküldi a benne foglalt kódot a böngészőhöz és beszünteti a lap további kiértékelését.

A `Buffer` tulajdonság, valamint a `Flush`, `Clear` és `End` metódus alkalmazását a `Response\Response.Buffer_vbs.asp` példalap mutatja be.

A `Redirect` metódus átirányítja a lapot egy másik lapra. Szintaxisa

```
Response.Redirect virtuális_erlérési_út
```

Amikor a `Server` objektumot tárgyaltuk, láttuk, hogy a `Server.Transfer` és a `Server.Execute` metódusok is átirányítják az aktuális lapról a böngészőt egy másik lapra.

Az egyik különbség a `Server.Transfer` és `Server.Execute` metódusok és a `Response.Redirect` metódus között abban áll, hogy az előbbiek akkor is alkalmazhatók, ha a böngészőhöz kódrészletet küldtünk, míg a `Response.Redirect` metódust csak akkor használhatjuk, ha még nem küldtünk kódot a böngészőnek.

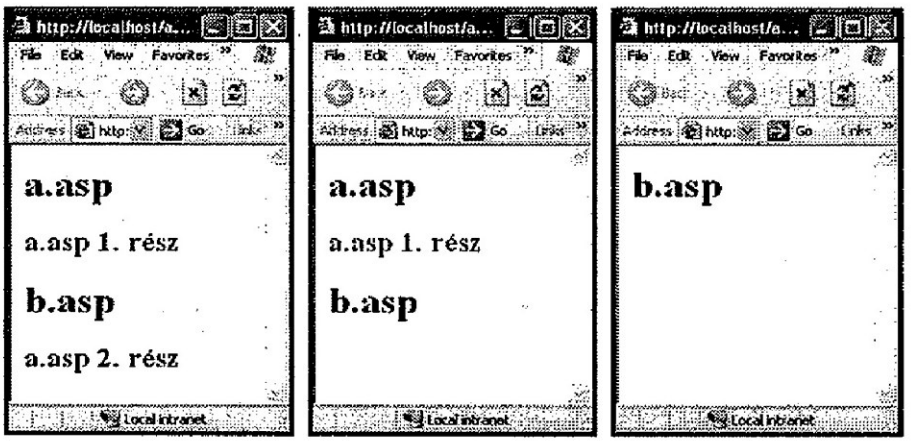
Mint tudjuk, a `Server.Transfer` és `Server.Execute` metódusok meghívása előtt levő kodrészlet megjelenik a böngészőben, majd a megjelenítés folytatódik a `Server.Transfer` vagy `Server.Execute` argumentumába megadott lap megjelenítésével. A `Server.Transfer` metódus esetében az első lap hátramaradt része nem lesz megjelenítve, míg a `Server.Execute` metódus estében a megjelenítés az első lap hátramaradt részének a megjelenítésével folytatódik.

Végezzünk itt is el egy kis kísérletet. Jelenítsük meg a következő `a.asp` lapot:

```
<%Response.Buffer=False%>
<HTML>
<BODY>
<H1>a.asp</H1>
<H2>a.asp 1. rész</h2>
<%
'Response.Flush
'Response.Clear
Server.Execute "b.asp"
'Server.Transfer "b.asp"
'Response.Redirect "b.asp"
%>
<H2>a.asp 2. rész</H2>
</BODY>
</HTML>
```

ahol a `b.asp` lap kódja a következő:

```
<html><h1>b.asp</h1></html>
```



2.27. ábra A `Server.Execute`, `Server.Transfer` *és* `Response.Redirect` *metódusok közti különbség*

Az első utasítás révén a pufferelés ki van kapcsolva, tehát a böngésző folyamatosan kapja a kódot a szervertől. A lapot a 2.27. ábra bal oldali képén láthatjuk. A `Server.Execute "b.asp"` utasítás előtt levő kódot az ASP elküldte a böngésző-

nek, ezután az említett utasítás hatására a b.asp lap kódja és végül az a.asp lap hátramaradt kódja került a böngészőhöz.

Most módosítsuk az a.asp lap kódját és a Server.Execute "b.asp" utasítás helyett hívjuk meg a Server.Transfer "b.asp" utasítást. Az eredményt a 2.27. ábra középső képén látjuk. A Server.Transfer "b.asp" utasítás előtt levő kódot az ASP elküldte a böngészőnek, majd a b.asp kódjával folytatta és lezárta a kód továbbküldését. Ezért az a.asp lap második része már nem jelenik meg.

Módosítsuk ismét az a.asp lapot és a Server.Transfer "b.asp" utasítást helyettesítsük a Response.Redirect "b.asp" utasítással. Ha az így módosított lapot letöltjük, akkor egy hibát idézünk elő. Ha tovább módosítjuk a lapot és bekapcsoljuk a pufferelést (Response.Buffer=**True**), akkor a 2.27. ábra jobb oldali képén látható megjelenítést kapjuk. Ezután hívjuk meg a Response.Redirect "b.asp" utasítás előtt a Response.Flush metódust! Ismét hiba! Végül helyettesítsük a Flush metódus a Clear metódussal! Újból megjelenik a jobb oldali képen látható lap.

Nézzük meg, hogy magyarázható mindez? Ebben az utolsó lépésben az a.asp lap négy változatát próbáltuk ki. Legelőször, mivelhogy a pufferelés ki van kapcsolva, ASP folyamatosan küldi a kódot a böngészőhöz, így mielőtt a Response.Redirect "b.asp" utasítás végrehajtásra kerül, a böngésző már megkapta a HTTP fejlécet. Most a Response.Redirect még egy fejlécet küld a böngészőhöz anélkül, hogy a böngésző megkapta volna az üzenetet, miszerint a megkezdett lap megjelenítése befejeződött. Ez viszont tiltott és ezt hozza tudomásunkra a böngésző egy hibajelzéssel. A lap második változatával bekapcsoltuk a pufferelést, tehát ASP beírja a HTTP fejlécet és az a.asp lap első részét a pufferbe, de egyelőre semmit se küld tovább. Következik a Redirect metódus meghívása. Ennek hatására ASP kiüríti teljesen a puffert, a HTTP fejlécet is, és beírja az új HTTP fejlécet és a b.asp lap kódját. Ezután ASP elküldi a puffer tartalmát a böngészőnek és beszünteti az a.asp lap további értelmezését: az a.asp lapból semmit sem látunk, csak a b.asp lapot jeleníti meg a böngésző. A következő változat beiktatja a Flush metódust közvetlen a Redirect metódus elé. A pufferbe felgyűlt adatok, így a HTTP fejléc is, egyszerre lesz elküldve a böngészőhöz. Majd a Redirect metódus kiüríti az egyébként üres puffert és beolvassa az új HTTP fejlécet és a b.asp lapot. Következő lépésként ASP elküldi a puffer tartalmát a böngészőnek, mely viszont már kapott egy HTTP fejlécet. Az ismert hiba megint bekövetkezik. Végül vizsgáljuk meg az a.asp lap utolsó változatát! Most a Flush metódus helyett a Clear metódust hívtuk meg. A Clear metódus meghívása előtt a puffer a HTTP fejlécet és az a.asp lap első részét tartalmazza. A Clear metódus kiüríti a puffert, de benne hagyja a HTTP fejlécet. A böngészőhöz még nem jutott el semmi. Következik a Redirect metódus. A puffer teljesen ki lesz ürítve, a benne levő HTTP fejléc is kikerül. Helyére az új HTTP fejléc és a b.asp lap kódja kerül.

ASP leküldi a puffer tartalmát a böngészőhöz és beszünteti az `a.asp` lap értelmezését. A böngésző ezúttal csak egy HTTP fejlécet és a `b.asp` kódját kapta.

Ez a kis kísérlet arra szolgál, hogy közelebbről megismerjük a `Redirect` metódus helyes használatát. Általános szabályként elmondhatjuk, hogy a `Redirect` metódust nem szabad meghívnunk, miután ASP valamit, legyen ez akár csak a HTTP fejléc is, elküldött a böngészőhöz.

Egy másik fontos különbség a `Server.Execute` és a `Server.Transfer` metódusok és a `Response.Redirect` metódus között abban áll, hogy a `Redirect` metódus argumentumában a lehívott lap virtuális elérési útjához lekérdező láncot (*query string*) is kapcsolhatunk, ami az előbbi két metódus esetében lehetetlen:

```
Response.Redirect _
"maslap.asp?nev1=ertek1&nev2=ertek2&nev3=ertek3"
```

Ez lehetővé teszi, hogy a `Redirect` metódussal letöltött lapot, a lekérdezési láncban foglalt *név = érték* pártoktól függően, dinamikusan hozzuk létre.

Ezt az eljárást illusztrálja a `Response\Response.Redirect_vbs.asp` lap.

Az `AddHeader` eljárással az előző szakaszban találkoztunk. Akkor megemlítettük, hogy segítségével úgynevezett felhasználói szerverváltozókat definiálhatunk. Szintaxisa

```
Response.AddHeader név, érték
```

ahol *név* az új szerverváltozó neve és *érték* a hozzárendelt érték. Ha például a `Prefered-Size` változót `"middle"` értékkel kívánjuk definiálni, akkor a metódust így hívjuk meg:

```
Response.AddHeader "Prefered-Size", "middle"
```

Ha egy lapon egy felhasználói szerverváltozót definiáltunk, akkor a lapon belül már nem távolíthatjuk el és egy újabb definíció esetleg más értékkel nem bírálhatja felül a változó legelső definícióját.

Láttuk, hogy az így meghatározott felhasználói szerverváltozókat lekérdezhetjük a `Request.ServerVariables` gyűjtemény segítségével:

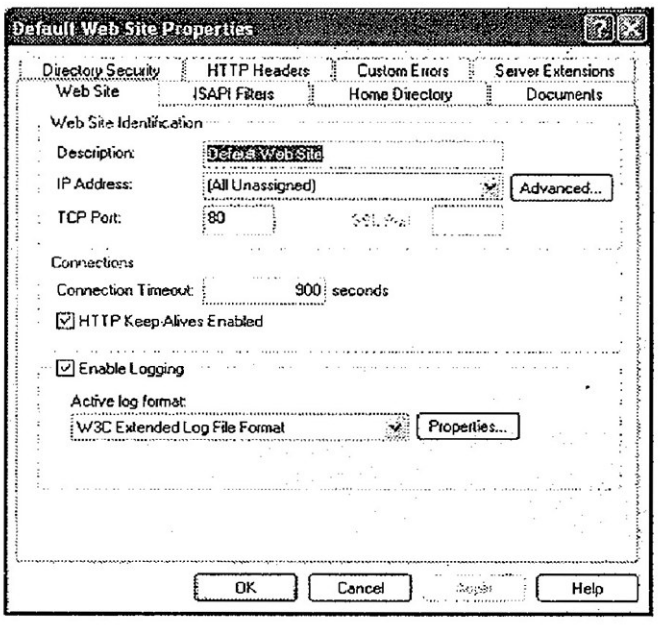
```
size = Request.ServerVariables("HTTP_Prefered_Size")
```

A lekérdezéskor a változó neve elé kell illessztenünk a `HTTP_` karaktercsoportot és mindegyik kötőjelet a _ karakterrel kell helyettesítenünk. Ezért tanácsos, hogy a változó eredeti nevében kerüljük a _ karakter.

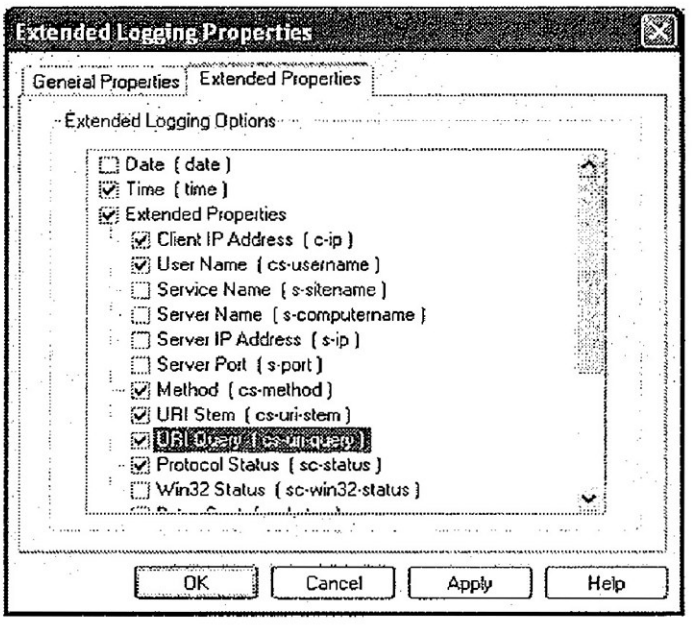
Az `AddHeader` metódust csak akkor szabad meghívni, amikor ASP még nem küldött el semmit a böngészőnek.

Az `AppendToLog` metódus segítségével egy karakterláncot írhatunk a webszerver naplójába.

Ennek elérésére webszerverünkön bizonyos beállításokat kell biztosítanunk. Az `AppendToLog` csak akkor írhat a naplóba, ha a napló formátuma *W3C Extended Log File Format* és a napló *URI Query* tulajdonsága be van állítva. Ezeket a kérelmeket így állítjuk be. Az IIS konzolban kattintsunk az egér jobb gombjával a szerverünkre és a megjelenő kontextusmenüből válasszuk ki **Properties** menüpontot! A megjelenő ablakban beállíthatjuk a napló tulajdonságait. Az **Activate protocol** jelölőnégyzettel aktiváljuk a naplót, az alatta levő lenyíló listában pedig állítsuk be a protokollformátumot *W3C Extended*-re (2.28. ábra)! Ezután nyomjuk meg a lenyíló lista mellett levő **Properties...** gombot! A most megjelenő ablakban a napló tulajdonságait szabhatjuk meg. Kattintsunk az **Extended Properties** fülre és jelöljük meg az **URI Queri (cs-uri-query)** jelölőnégyzetet majd nyomjuk meg az **OK** gombot (2.29. ábra)! Visszatérve az első ablakra nyomjuk meg az **Apply** gombot.



2.28. ábra Beállítjuk a napló formátumát



2.29. ábra Beállítjuk a napló tulajdonságait

Most mindent előkészítettünk, hogy a naplóba írhassunk. Az `AppendToLog` metódus szintaxisa egyszerű:

```
Response.AppendToLog karakterlánc
```

A paraméterként megadott *karakterlánc* változóban tárolt karakterláncot a szerver hozzáfűzi a naplóba írt szöveghez. Alapértelmezésben a napló a `WINNT\System32\LogFiles\W3SVCn` mappába van, ahol *n* a szerveren levő webhelyek száma. Az alábbi kódrészlettel

```
<%
strToAppend= _
"Ezt a sort az AppendToLog metódussal írtuk"
Response.AppendToLog strToAppend
Response.Write "Beírtunk a naplóba!"
%>
```

a következő sort írjuk a naplóba:

```
11:15:32 127.0.0.1 - GET
/aspclassic/VBS/2.+fejezet/Response/appendtolog.asp
Ezt+a+sort+az+AppendToLog+metódussal+írtuk 200
```

A szóközökkel elválasztott adatmezők jelentését a napló elején olvashatjuk:

```
#Fields: time c-ip cs-username cs-method cs-uri-stem cs-uri-query
sc-status
```

Ha hátramaradt két metódust csak röviden említem meg.

A `BinaryWrite` metódus segítségével egy tömb tartalmát binárisan beleírhatjuk a böngészőhöz vagy a szerverhez küldött adatsorba. A metódust például akkor használjuk, amikor a szerveren elhelyezett adatbázisban tárolt képeket akarunk megjeleníteni, vagy amikor állományokat kívánunk a szerverre felküldeni. Példákat a későbbi fejezetekben fogunk látni.

A `PICS` metódus lehetővé teszi, hogy bizonyos értelemben a letöltendő webalkalmazás tartalmáról a kliens még a letöltés előtt információt szerezzen, és ennek ismeretébe eldöntheti, hogy kívánja-e az alkalmazást letölteni vagy nem. A PICS (*Platform for Internet Control Selection*) specifikációt 1995-ben javasolták. A specifikáció a filmeknél és a videóknál használt címkézés mintájára 0-tól 4-ig terjedő fokozattal rendelkező címkével lát el egy alkalmazást, amit a felhasználó lekérdezhet. A címkézés négy kategóriába történik: *language*, *nudily*, *sex* és *violence*. Az **Internet Explorer**-ben a felhasználó beállíthatja magának, hogy melyik kategóriába, milyen fokozattal tiltja a lapok letöltését. Az **Internet Explorer 6.x**-ben a **Tools/Internet Options...** menüpontból kell elindulnunk és a megjelenő ablakon **Content** fület kell előtérbe hoznunk. A **Content Advisor** szakaszban az **Enable** gomb lenyomásával a megjelenő ablak **Ratings** fülére kattintva eljutunk a beállításokhoz, illetve itt módosíthatjuk.

Ezek a beállítások csak akkor hatnak, ha a letöltendő lapok tartalmazzák a PICS címkét. A címeket a webhelyek kérésre, egy erre a célra létrehozott hatóságtól, melyeket civil szervezetek működtetnek, kapják. A legismertebb ilyen hatóságok a RSACi (http://www.rsac.org/) és a SafeSurf (http://www.classify.org/safesurf/). Bővebb információt a PICS-ről ezeknek a szervezeteknek a honlapjáról tudhatunk meg.

Most térjünk át a `Respons` objektum tulajdonságaira!

2.6.2 A `Response` objektum tulajdonságai

A `Buffer` tulajdonsággal az előbbi alszakaszban foglalkoztunk és láttuk szerepét a `Flush`, `Clear` és `End` metódusokkal egybekötve.

A legtöbb esetben a kliensgép nincs egyenes kapcsolatban a szerverrel. Közöttük egy „megbízott" szerver, egy proxy helyezkedik el, mely mint egy átjáró (*gateway*) működik. A proxy szerepe, hogy elválassza a szervert működtető vállalat vagy szervezet hálózatát a külső hálózattól, növelve a kényes adatok biztonságát. A hálózat tehermentesítése érdekében a proxy arra is használható, hogy a gyakran kért weblapokat tárolja és a forrásszerver helyett, ismételt kérésre, rendelkezésre bocsátja. Ezt az eljárást *cashing*-nek vagy gyorskiszolgálásnak szokták nevezni és ebben az esetben a proxy, más szerepe mellett, egy gyorskiszolgáló szerver szerepét is átveszi. A gyorskiszolgáló szerver előnye, hogy a hálózatban könnyen elérhető, tehát az adatokat nagyon gyorsan adja ki. Hátránya az, hogy a kiszolgált adatok nem feltétlenül az aktuális állapotot tükrözik, hanem az eltárolásuk pillanatában fennállt állapotot adják vissza.

A `Response` objektum `CashControl` tulajdonságával lekérdezhetjük, illetve megszabhatjuk a gyorskiszolgálás módját. Szintaxisa a következő:

```
str = Response.CashControl
Response.CashControl = "Public" | "Private"
```

Alapértelmezett értéke `"Private"`, mely nem engedi meg a lap proxyn való tárolását. Ez abban az esetben fontos, amikor a lapon megjelenő adatok gyorsan változnak (például tőzsde értékek) vagy ha a lapot a felhasználó bizonyos korlátok között (háttérszín, menüpontok stb. beállítása), saját igénye szerint beállíthatja. Ilyenkor mindig a legfrissebb adatokat kell szolgáltatni és ebben az esetben a gyorskiszolgáló szerver csökkenti a lap dinamikus voltát. Ha viszont olyan lapokról van szó, melyek ritkán változnak, esetleg ritkán változó nagy adatmennyiséget tartalmaznak, akkor a gyorskiszolgáló bevetése célszerű és kívánatos. Ilyenkor a `CashControl` tulajdonságot `"Public"` értékre állítjuk be.

A proxyn tárolt lapok természetüknél fogva valamikor elavulnak, tehát újítani kell tartalmukat, azaz jeleznünk kell, hogy mennyi idő eltelte után a lap jelenlegi formájában már nem aktuális. Egy lap élethosszát a `Response` objektum `Expires`, illetve `ExpiresAbsolute` tulajdonságával szabályozhatjuk.

Az `Expires` tulajdonság visszaadja, illetve megszabja hány percig érvényes a lap:

```
lMinutes = Response.Expires
```

illetve

```
Response.Expires = lngMinutes
```

ahol `lngMinutes` a lap érvényességének időtartama percekben. Azért, hogy egy lap azonnal érvényességét veszítse, adjunk a tulajdonságnak egy nagy negatív értéket, például `-10000`.

Az `ExpiresAbsolute` egy dátumot ad vissza, illetve határoz meg, amelyik után a lap már nem érvényes. A visszaadott, illetve a tulajdonságot megszabó érték egy dátumot és egy időpontot megadó karakterlánc.

Ha a látogató az `Expires`, illetve `ExpiresAbsolut` tulajdonság által megadott időpont előtt tölti le a lapot, akkor a lap a `CashControl` tulajdonág által megadott gyorsító tárolóból lesz letöltve. Ha `CashControl` értéke `"Private"`, akkor a felhasználó gyorsító tárolójából, ha `CashControl` értéke `"Public"`, akkor a proxy gyorsító tárolójából lesz a lap letöltve. Ha lap érvényessége lejárta után hívjuk le, akkor a forrásszerverről lesz letöltve.

A `Charset` tulajdonsággal lekérdezhetjük a beállított, illetve beállíthatjuk a lapon használt karakterkészletet:

```
strKKeszlet = Response.Charset
Response.Charset = charset_string
```

ahol *`charset_string`* a karakterkészletet leíró karakterlánc.

Mostanáig láttuk, hogy a karakterkészletet a

```
<%@ CODEPAGE="code_page_number" %>
```

rendelkezéssel, vagy a `Session.CodePage` tulajdonságával szabhattuk meg. A `Response.Charset` tulajdonsággal megadott karakterkészlet felülbírálja a `CODEPAGE` rendelkezéssel, illetve a `Session.CodePage` tulajdonsággal megadott karakterkészletet, de csak az adott lapon belül.

A böngészőnek „tudnia" kell, mit kap a szervertől megjelenítésre: egy HTML lapot, egy képet, egy audió vagy video állományt stb. A HTTP tartalmat a MIME típus írja le. MIME a *Multi-purpose Internet Multimedia Extension* vagy *Multi-purpose Internet Mail Extension* angol kifejezések rövidítése. A MIME típusokat standardizált karakterláncok adják meg "*típus/altípus*" formában. A legismertebbek közé tartoznak a következők:

MIME típus	leírás	kiterjesztés
"text/html"	HTML állomány	.htm, .html
"text/xml"	XML állomány	.xml
"image/gif"	GIF formátumú képállomány	.gif
"image/jpeg"	JPG formátumú képállomány	.jpeg, .jpg, .jpe
"application/msword"	MS Word dokumentum	.doc
"application/msaccess"	MS Access dokumentum	.mdb
"application/pdf"	PDF (Adobe Acrobat) dokumentum	.pdf
"video/x-msvideo"	MS Windows Video állomány	.avi
"video/x-sgi-movie"	SGI Movieplayer állomány	.movie

A ftp://ftp.isi.edu/in-notes/iana/assignments/media-types címről letölthetjük a jelenleg ismert MIME típusok listáját.

A böngésző kétféleképpen értesülhet a meglátogatott állomány MIME típusáról. Ha a felhasználó a böngésző segítségével letölt egy állományt, a böngésző az állomány kiterjesztése alapján állapítja meg az állomány MIME típusát és felkészül a megfelelő kezelésére. Ha például a böngésző egy video állományt kér, akkor elindíthatja a megfelelő kezelőprogramot (feltéve, hogy a rendszer rendelkezik ilyennel). A második lehetőség a böngészőnek tudtára adni, hogy mi vár rá, az ASP segítségével történhet.

A `Response.ContentType` tulajdonság kiolvassa, illetve megszabja a böngészőnek szánt MIME típust:

```
strMIMEType = Response.ContentType
Response.ContentType content_type
```

ahol *content_type* a MIME típust azonosító karakterlánc.

Amikor egy lapot a böngésző letöltött, a szerver jelzi a böngészőnek, hogy a letöltés hibamentesen vagy esetleg hibákkal zajlott le. A jelentés szövegét a `Response.Status` metódussal kérdezhetjük le, illetve adhatjuk meg:

```
strSatuts = Response.Status
```

illetve

```
Response.Status strSatuts
```

Ha bonyolultabb, sok időt igénybevevő műveletek előtt lapunk látogatója nem várja meg a lap megjelenítését és átkapcsol más lapra, erőforrás pazarlásnak számít a bonyolult művelet végrehajtása. Ilyenkor jó lenne az időigénylő műveletet abbahagyni, ezzel is hozzájárulva a szerver túlterhelésének az elkerüléséhez. Ehhez viszont tudnunk kell, hogy a látogató várja-e még a lap letöltését, vagy egy másik lapot kért közben.

Az `IsClientConnected` tulajdonság **True** értéket ad vissza, ha a lap látogatója a lekérdezés pillanatában még mindig kapcsolatban áll a szerverrel és nem kért közben más lapot. Ellenkező esetben a tulajdonságnak az értéke **False**. A következő kódrészlet bemutatja az `IsClientConnected` tulajdonság használatát:

```
If Not Response.IsConnected Then
  Response.End
End If
```

Ezzel befejeztük a `Response` objektum metódusainak és tulajdonságainak áttekintését. Hátramaradt a `Cookies` gyűjtemény, amit a `Request` objektum `Cookies` gyűjteményével együtt a következő alszakaszban fogunk tárgyalni.

2.7 A `Response` és `Request` objektum `Cookies` gyűjteménye

A `Cookies` gyűjteménybe tárolt objektumok angol elnevezése *cookie* és pillanatnyilag még nincs egységesen elfogadott magyar megfelelője. Használatban vannak a süti, kuki vagy keksz elnevezések. Én a süti elnevezést fogom használni.

A süti a felhasználó gépen tárolt információ, mely az azt létrehozó szerver számára fontos információt tartalmaz a felhasználóról vagy a gépéről. A sütikkel kapcsolatban sok legenda forog az internetező közönségben közszájon, ezért mindjárt az elején jó lenne egy pár dolgot tisztázni.

A süti egy kis szövegállomány, amit a látogató böngészője ír a látogató merevlemezére, ha ezt a meglátogatott lap forráskódja kéri, és ha a látogató úgy állította be a böngészőjét, hogy engedélyezi a sütik tárolását. Mit tartalmazhat egy süti? Elsősorban olyan adatokat, amiről a szervernek tudomása van. Ezek az adatok a szerverre vagy a kliensre vonatkozhatnak. Ilyen adat a szerver *domain* neve vagy az időpont, amikor a látogatás megtörtént, de a böngésző típusa vagy a kliensgép operációs rendszere is ide tartozik. Elméletileg a szerver a `Request.ServerVariable` bármely értékét beírhatja egy sütibe, habár ennek nincs sok értelme. Másodsorban olyan adatok kerülhet-

nek a sütibe, melyeket mi magunk, mint látogatók közöltünk a szerverrel. Bekerülhet tehát egy sütibe a telefonszámunk, a rendszerbe való belépésre szükséges jelszó, vagy – Isten ments! – a folyószámlánk száma? A válasz nagyon egyszerű: ha ilyen adatokat megadtunk a szervernek, akkor igen, ha viszont bizalmas adatot nem közöltünk a szerverrel, akkor természetesen nem tudhat róla és nincs mód arra, hogy egy süti elhelyezésével a gépünkre a szerver ilyen adatok birtokába jusson.

Miután ezeket tisztáztuk, nézzük meg, hogy írhatunk sütiket a látogató gépére! Erre a célra a `Response` objektum `Cookies` gyűjteményét használjuk:

```
Response.Cookies(süti_név)[(kulcs)] = érték
```

Itt *süti_név* és *kulcs* karakterláncok. A *kulcs* paraméter megadása opcionális. Ilyenkor egyértékű sütiről beszélünk. Ha viszont a *süti_név* sütit több *kulcs* paraméterrel definiáljuk, akkor többértékű sütiről beszélünk. A fenti utasítás elküldi a böngészőnek a létrehozott sütit a HTTP fejléccel együtt. Ezért, a fenti utasítást nem szabad használnunk, ha a böngészőhöz már elküldtünk kódrészletet. Ha a `Response.Buffer` tulajdonságnak az alapértelmezett **`True`** értékét nem változtatjuk meg, akkor a lap folyamán, addig ameddig a `Response.Flush` metódust nem hívtuk meg, bárhol létrehozhatunk sütit.

A böngészőhöz elküldött süti bekerül a böngésző számára fenntartott memóriaszeletbe. A süti csak akkor kerül a lemezre, ha a szerver, azaz a meglátogatott ASP lap ezt kifejezetten kéri (és természetesen a böngésző ezt nem tiltja). Erről később többet fogunk megtudni a szakasz folyamán. Egyelőre tételezzük fel, hogy a böngésző befogadja a sütiket.

Ha egy süti létrehozásakor már létezik egy azonos nevű süti, akkor az új értékek átírják a régieket.

A `Response.Cookies` gyűjteménynél is megtaláljuk a gyűjteményeknél szokásos tulajdonságokat: `Count`, `Item` és `Key`. A gyűjtemény tagjai maguk is objektumok és tulajdonságokkal rendelkeznek. A `Response.Cookies` gyűjteménybe foglalt sütik tulajdonságai a következők:

- `Domain`
- `Expires`
- `HasKeys`
- `Item`
- `Path`
- `Secure`

A `Domain` tulajdonság megszabja, melyik *domain*-en belül látható a süti. Az alapértelmezett értéke a sütit létrehozó lap *domain*-e. Ha egy süti értékeit egy más *domain*-

ből, mint a süti `Domain` tulajdonságban megadott *domain* kérdezzük le, akkor a süti nem látható. A `Domain` nem olvasható, csak az értékét szabhatjuk meg:

```
Response.Cookies("myCookie").Domain = "http://www.myhost.com"
```

Az `Expires` tulajdonsággal megszabhatjuk a süti érvényességét. A tulajdonság értékét egy `Date` típusú adatként kell megadni:

```
' ez a süti egy hétig érvényes
Response.Cookies("myCookie").Expires = DateAdd("d", 7, Now)
```

Ezt a tulajdonságot sem kérdezhetjük le.

Ha egy süti `Expires` tulajdonságát kifejezetten nem adjuk meg, akkor a süti nem kerül a merevlemezre. A böngésző bezárásával és a számára fenntartott memóriarész kiürítésével megszűnik. Ha viszont megadjuk a süti `Expires` tulajdonságát, akkor a böngésző, amennyiben ezt a beállításánál nem tiltottuk meg, a merevlemezre ír egy kis szövegállományt, mely tartalmazza a sütit. Később közelebbről is megnézünk egy ilyen süti-állományt.

Sokszor szükséges tudnunk, hogy egy süti egy- vagy többértékű. Ez különösen akkor fontos, ha egy süti értékét le akarjuk kérdezi. A `HasKeys` tulajdonság értéke **`True`**, ha a sütinek több értéke van, illetve **`False`**, ha a süti egyértékű. A tulajdonságot nem változtathatjuk meg, csupán értékét kérdezhetjük le. A következő kódrészlet kitörli a `"myCookie"` süti összes értékét:

```
Set cookie = Response.Cookie("myCookie")
If cookie.HasKay Then
        For Each key in cookie
              cookie(key)= ""
        Next
Else
        cookie=""
End If
```

Az `Item` tulajdonság visszaadja az argumentumával megadott kulcs vagy név értékét:

```
strUid = Response.Cookie("myCookie").Item("userid")
```

A `Path` tulajdonság egy mappa virtuális elérési útját várja:

```
Response.Cookie("myCookie").Path = "/aspclassic/VBS"
```

Alapértelmezése az alkalmazás törzsmappája. Ezzel a tulajdonsággal, amit megadhatunk, de nem kérdezhetünk le, megszabjuk, hogy mely lapok számára látható a süti. A megadott mappán kívül levő lapokról a sütit nem lehet elérni.

A süti utolsó tulajdonsága a `Secure` tulajdonság. Értéke **`True`** vagy **`False`** lehet és nincs módunk lekérdezni:

```
Response.Cookie("myCookie").Secure = True | False
```

Ha tulajdonság értéke **True**, akkor a süti csak HTTPS protokollal elérhető lapról látható. Ellenkező esetben bármilyen protokollal elérhető lapról elérhető.

Térjünk most át a `Request` objektum `Cookies` gyűjteményére.

Ez a gyűjtemény is a megszokott három tulajdonsággal rendelkezik: `Count`, `Item` és `Key`. A gyűjtemény segítségével kérdéseket küldhetünk a böngészőnek a kliensgépen levő sütikről:

```
str = Request.Cookies(azonosító [, elem])
```

Az első argumentum lehet a süti neve vagy a süti sorszáma a gyűjteményben. A második argumentum opcionális és többértékű sütik esetében megadja a lekérdezett elem nevét.

A `Request.Cookies` gyűjtemény tagjai, úgy mint a `Response.Cookies` gyűjtemény esetében, objektumok, de természetesen más típusúak. Kevesebb tulajdonsággal rendelkeznek, mint a `Response.Cookies` gyűjtemény tagjai:

- `Count`
- `HasKey`
- `Item`
- `Key`

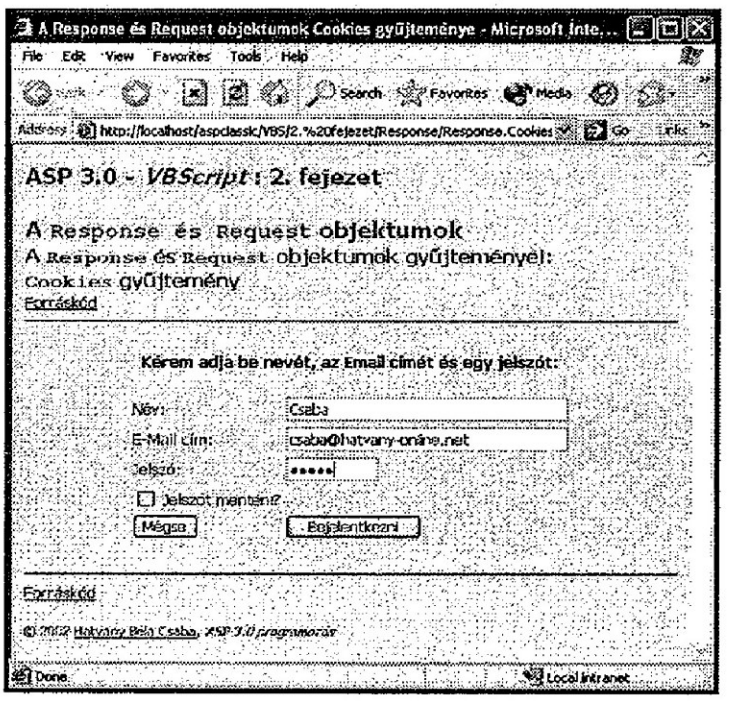
A `HasKey` és az `Item` tulajdonságok megegyeznek a `Response.Cookies` gyűjtemény tagjainak azonos nevű tulajdonságaival. A `Count` tulajdonság visszaadja a süti elemeinek számát, míg a `Key` tulajdonság a sütielem nevét vagy kulcsát adja meg.

A következő kódrészlet kiírja a "`myCookie`" nevű süti elemeinek nevét és értékét:

```
Set cookie = Request.Cookies(("myCookie")
For intCount = 1 To cookie.Count
        Response.Write cookie.Key(intCount) & " = " & _
            cookie.Item(cookie.Key(intCount)) & "<br>
Next
```

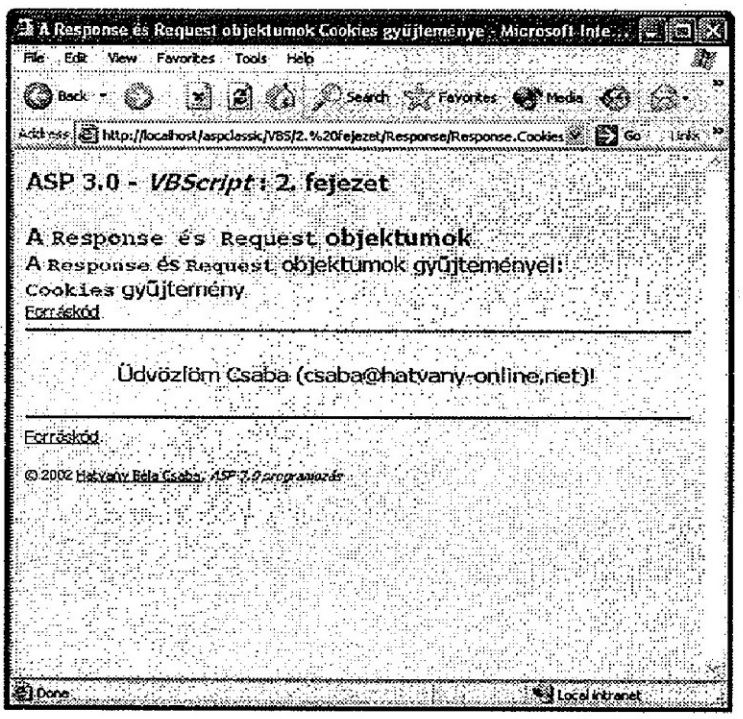
Befejezésül, vizsgáljuk meg a `Response/Response.Cookies_vbs.asp` példát. Ez a példa, leegyszerűsített formában, megmutatja, hogy használhatjuk fel a sütiket a látogatók felismerésére. Biztosra vehetem, hogy a kedves Olvasó találkozott már olyan honlappal, ahova egy névvel és egy jelszóval be kellett jelentkeznie. Biztos már látott olyan esetet is, amikor lehetősége volt arra, hogy eldöntse, akarja-e a jelszót menteni? Ha kiválasztotta ezt a lehetőséget, a lap legközelebbi látogatásakor már nem kellett

bejelentkeznie, mert az alkalmazás felismerte. Jelen példánk ugyanígy működik. Mielőtt a példa részletes tárgyalásába belekezdenénk, nézzük meg a példaalkalmazást! Töltsük le a `Response/Response.Cookies_vbs.asp` lapot (2.30. ábra).



2.30. ábra Egy bejelentkező lap, mely a sütik segítségével felismeri látogatóját

Töltsük ki a szükséges mezőket, de ne jelöljük ki a **Jelszót menteni?** jelölőnégyzetet, majd kattintsunk a **Bejelentkezni** gombra! Az alkalmazás az üdvözlő lappal hozza tudomásunkra, hogy sikeresen bejelentkeztünk (2.31. ábra).



2.31. ábra A bejelentkezés sikerrel járt

Most csukjuk be a böngészőt és töltsük le újból a lapot. Az alkalmazás a már ismert bejelentkező lappal indul (2.30. ábra). Töltsük ki ismét a szükséges mezőket, ezúttal azonban, jelöljük ki a **Jelszót menteni?** jelölőnégyzetet, majd kattintsunk a **Bejelentkezni** gombra! Elvárásunknak megfelelően, most is sikeres volt a bejelentkezés. Végül csukjuk be a böngészőt és töltsük le utoljára a lapot. Ezúttal, minden bejelentkezés nélkül az alkalmazás az üdvözlő lappal köszönt.

A `Response/Response.Cookies_vbs.asp` lap a HTML vezérlőket egy `<FORM>` elemben csoportosítja. A `<FORM>` elem a beadott adatokat az `ACTION` attribútummal megadott `Response.Cookies/CheckCookie_vbs.asp` laphoz a `METHOD` attribútummal megadott `POST` metódussal küldi le:

```
<%
  Dim aspCookie
  Set aspCookie=Request.Cookies("aspCookie")

  If aspCookie.HasKeys Then
         Response.Redirect _
       "Response.Cookies/CheckCookie_vbs.asp?cookie=1"
  End If
%>
<CENTER>
<H3>Kérem adja be nevét, az Email címét és egy
jelszót:</H3>
<FORM ACTION = "Response.Cookies/CheckCookie_vbs.asp"
METHOD="POST" >
<!—itt folytatódik a forráskód -->
```

Mielőtt azonban a `<FORM>` megjelenik, ASP egy rövid kódrészletet értelmez és végrehajt. A `Request.Cookies` gyűjtemény segítségével a lap lekérdezi az `"aspCookie"` nevű sütit. Ha a jelszót nem mentettük el, akkor egy ilyen nevű süti nem létezik, tehát az `aspCookie` süti objektum üres. Következésképpen a `HasKeys` tulajdonsága **`False`** értékű, azaz megjelenik a `<FORM>` elem a vezérlőkkel. Miután kitöltöttük a kérdőívet és az `Response.Cookies/CheckCookie_vbs.asp` lapnak elküldtük, ASP elkezdi ennek a lapnak ez értelmezését:

```
<%
Dim strEMail
Dim strUid
Dim strPwd
  ' ha létezik a süti, kiolvassuk az adatokat
  If Request.QueryString("cookie")=1 Then
    strEMail=Request.Cookies("aspCookie")("email")
    strUid=Request.Cookies("aspCookie")("uid")
    Response.Write "<CENTER><H2>Üdvözlöm " & _
          strUid & " (" & strEmail & ")!</H2></CENTER"
  ' különben az adatokat a Form gyűjteményből
  ' olvassuk ki
```

```
Else
  strEmail=Request.Form("Email")
  strUid=Request.Form("UserId")
  strPwd=Request.Form("Password")
  If strEmail="" Or strUid="" Or strPwd="" Then
     Response.Write _
              "<CENTER><H2>" & _
              "<Font color=red>" & _
              "Mindegyik adatot be kell adni! " & _
              "</Font></H2><BR>"
     Response.Write _
              "<A HREF=" & _
              """../Response.Cookies_vbs.asp""" & _
              ">VISSZA</A></CENTER>"
  Else
     ' ha menteni kell a jelszót, akkor
     If Request.Form("SaveLogin") = "on"  Then
        ' kiolvassuk az adatokat a Form
        ' gyűjteményből és kiírjuk egy sütibe
        Response.Cookies("aspCookie")("uid") = _
             Request.Form("UserID")
        Response.Cookies("aspCookie")("email") = _
             Request.Form("email")
        Response.Cookies("aspCookie")("pwd") = _
             Request.Form("password")
        Response.Cookies("aspCookie").Expires = _
             Date + 7
      End If
     Response.Write _
              "<CENTER><H2>Üdvözlöm " & _
              strUid & _
              " (" & strEmail & ")!</H2></CENTER"
  End If
End If
%>
```

Ha a jelszót nem mentettük el, akkor a `Form` gyűjteményben nem szerepel a `"SaveLogin"` név és a `Request.QueryString` gyűjteményben nem szerepel a `"cookie"` név. Most már csak azt kell ellenőrizni, hogy a látogató kitöltött-e minden szükséges mezőt és ha igen, akkor megjeleníthetjük az üdvözlő szöveget. Ez esetben az üdvözlő szövegben szereplő nevet és email címet a `From` gyűjteményből olvassuk ki.

Ha mentettük a jelszót, akkor a `Form` gyűjteményben a `"SaveLogin"` név `"on"` értékkel szerepel és a `Request.QueryString` gyűjteményben ismét nem szerepel a `"cookie"` név. Ebben az esetben, ha a látogató a kérdőív minden kérdésére felelt, létrehozunk egy `"aspCookie"` nevű többértékű sütit. A süti értékeit a `Form` gyűjteményből nyerjük.

Hol van ez a süti a merevlemezen? Amikor az **Internet Explorer**-t a felhasználó a gépére telepítette, akkor a telepítő program a felhasználó törzsmappájában létrehozta a `Cookies` almappát. Itt kell keresnünk a sütiket. Nézzük meg ennek a mappának a tartalmát! A mappában legalább egy szövegállományt találunk, a most elmentett sütit. A böngésző a következő séma szerint nevezi el ezeket a kis szövegállományokat:

felhasználónév`@`*alkalmazásnév*`[`*n*`].txt`

ahol *felhasználónév* a felhasználó bejelentkezési neve a rendszerbe, *alkalmazásnév* a webalkalmazás neve és *n* egy egész szám. Nyissuk ki a sütit képező állományt! Ha a **Nodepad**-et használjuk, akkor egy hosszú sort láttunk, mivelhogy a sorok vége nem a szokásos `
& #10;` (*carriage return*, *linefeed*) karakterpárral, hanem csak a `
` (*linefeed*) karakterrel van megjelölve. Előnyösebb tehát, ha az állományt egy másik szövegszerkesztővel, például a már említett, **TextPad**-el nézzük meg. Az én esetemben az állomány neve `csaba@VBS[1].txt` és ez a tartalma:

```
aspCookie
pwd=engedjbe&email=csaba%40hatvany%2Donline%2Enet&uid=Csaba
localhost/aspclassic/VBS
1024
3187070976
29523669
2863473120
29522405
*
```

Az állománynévben felismerjük a felhasználónevet (`csaba`) és az alkalmazásnevet (`VBS`). Az állomány első sora a süti nevét tartalmazza: `aspCookie`. A második sor egy lekérdezési láncra emlékeztet. Ha figyelmesen megvizsgáljuk, URL kódokat ismerünk fel: `%40`, `%2D` és `%2E` sorra `@`,- (kötőjel), illetve . (pont) karaktereknek felelnek meg. Ezzel a második sort így értelmezhetjük:

```
pwd=engedjbe&email=csaba@hatvany-online.net&uid=Csaba
```

Most tisztán láthatjuk a *név* = *érték* párokat. A nevek a süti kulcsai, az értékek a kulcsokhoz rendelt sütiértékek. Ezeket a `Request` objektum `Cookies` gyűjteményével kiolvashatjuk a sütiből. A harmadik sorban értelemszerűen felismerjük alkalmazásunk virtuális elérési útját: `localhost/aspclassic/VBS`. A többi sor kódolt adatokat tartalmaz a sütiről, többek között az érvényességének időtartamáról. Ezekre most nem térünk itt ki. A ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt cím alatt bővebb információ található a sütikről.

Térjünk vissza az alkalmazásunkhoz! Ha a böngésző még nyitva van, zárjuk be és hívjuk le újból a lapot. A lap elején levő ASP kódrészlet most is megvizsgálja, hogy van-e a `Request.Cookies` gyűjteményben egy `"aspCookie"` nevű süti. A válasz

most igenlő és a süti kulcsokkal rendelkezik. Következésképpen, a lap a `Response.Redirect` metódus segítségével, egy lekérdezési lánccal együtt, át lesz irányítva a `Response.Cookies/CheckCookie_vbs.asp?cookie=1` lapra.

A bejelentkező lap további kódját ASP nem veszi figyelembe. Ehelyett a megadott célt kezdi vizsgálni. Itt legelőször megvizsgálja a kapott lekérdezési láncot. Megtalálja a `cookie` nevet az `1` értékkel és most lekérdezi a süti adatait, azaz a nevet, amivel a felhasználó bejelentkezett és az email címet. Most már csak az üdvözlő szöveget kell megjeleníteni.

Ha a sütit még egy egészszámú értékkel láttuk volna el, melyet minden látogatás alkalmával eggyel megnövelünk (ebben az esetben minden látogatás alkalmával a sütit újra kellene írnunk), akkor a látogatót, amellett, hogy felismertük volna, arról is tájékozathattuk volna, hányadszor látogatja a lapot.

Mint látjuk, a sütik fontos információkat nyújthatnak a webalkalmazásoknak. A sütik alkalmazásakor ügyelnünk kell azonban arra, hogy alkalmazásuk akkor is hibátlanul működjön, ha a felhasználó úgy dönt, hogy nem fogad el sütiket.

2.8 Az `ASPError` objektum

Az ASP 3.0 a hibakeresés elősegítésére bevezeti az új `ASPError` objektumot. Az `ASPError` objektum segítségével a hibakeresés lényegesen egyszerűbbé válik, mint az ASP előző verzióiban. Az `ASPError` objektum információt szolgáltat a legutoljára bekövetkezett ASP hibáról. Ezt az információt felhasználhatjuk a bekövetkezett hiba kezelésére.

Az `ASPError` objektumot a `Server` objektum `GetLastError` metódusával hozzuk létre:

```
Set objASPError = Server.GetLastError()
```

Az így létrehozott objektum tulajdonságai tartalmazzák a bekövetkezett hibák leírását. Az `ASPError` objektumnak összesen kilenc tulajdonsága van:

- `ASPCode`
- `ASPDescription`
- `Category`
- `Column`
- `Description`
- `File`

- Line
- Number
- Source

Ezek a tulajdonságok a hiba milyenségéről, forrásáról tudósítanak, és ha ez lehetséges, a forráskódban is megmutatják a hiba helyét. Természetesen a tulajdonságokat nem módosíthatjuk, csupán lekérdezésük áll módunkban.

2.8.1 Az ASPError objektum tulajdonságai

Az ASPCode tulajdonság egy karakterláncot ad vissza, mely az ASP hibakódot tartalmazza. A karakterláncot az IIS generálja és legtöbbször az ASP szkriptben előforduló hibák esetében üres. Az ASPDescription az ASPCode hibakód magyarázatát vagy hibajelentését tartalmazza.

Leggyakrabban külső komponensek okozta hibáknál, illetve az IIS preprocesszora (*pre-processor*) által talált hibák esetében ad ez a két tulajdonság tájékoztatást. Az alábbi táblázat a leggyakrabban előforduló ilyen hibákat tartalmazza:

ASP hibakód (ASPCode)	ASP hibajelentés (ASPDescription)	Magyarázat
ASP 0100	Out of memory	A rendszer nem képes a szükséges memóriarészt biztosítani
ASP 0100	Unexpected error	A rendszer nem ismerte fel a hibát
ASP 0102	Expecting string input	A rendszer egy karakterláncot vár
ASP 0103	Expecting numeric input	A rendszer egy számot vár
ASP 0104	Operation not allowed	Tiltott művelet
ASP 0105	Index out of range	Az index a korláton kívül van
ASP 0106	Type mismatch	Nem megfelelő adattípus
ASP 0177	Server.CreateObject failed	Hibás PropgID
ASP 0241	CreateObject exception	Egy objektum létrehozása nem járt sikerrel

A Függelékben megtaláljuk az ASPError hibakódjainak teljes listáját.

A fejezetnek megfelelő mappában a ASPError1_vbs.asp és ASPError2_vbs.asp állományokkal idézhetünk elő ilyen hibát. Az első állomány hibásan alkalmazza az #include utasítást, amelyikkel bővebben a következő fejezetben foglalkozunk:

```
<!-- #include virtual="aspclassic/nonexistent.asp"-->
```

A helyes attribútum `virtual`. A második ASP állomány egy nem létező állományt jelöl meg, amit a preprocesszor nem talál meg és nem küldi az állományt tovább az ASP-hez, hanem hibát jelez:

```
<!-- #include virtual="aspclassic/nonexistent.asp"-->
```

Ha egy hiba keletkezik, akkor a `Category` tulajdonság megjelöli a hiba forrását. Arról értesülünk, hogy ASP hibáról, szkript hibáról vagy külső hibáról van szó. A `Category` tulajdonságtól visszaadott karakterlánc soha sem üres, ha egy hiba keletkezett.

Az előbbi példák ASP hibát idéznek elő. Ha a szkriptben előforduló hiba már a fordításkor kiszűrhető és így nem kerül sor az ASP lap *értelmezésére*, akkor egy fordítási hibáról beszélünk. Ha csak a futásidőben keletkezik, és emiatt megszakad az ASP lap *végrehajtása*, akkor futási hibáról van szó. Ezeket a hibákat az `ASPError` külön kategóriába sorolja.

Az `ASPError3_vbs.asp` és `ASPError4_vbs.asp` példaállományok fordítási hibákat, míg az `ASPError5_vbs.asp`, `ASPError6_vbs.asp`, `ASPError7_vbs.asp` és `ASPError8_vbs.asp` példaállományok futás közben történő hibákat hoznak létre.

A `Description` tulajdonság a keletkezett hiba rövid leírását tartalmazza, míg a `File` tulajdonság a hibát okozó állomány fizikális elérési útját adja vissza. Egy hiba esetén ezek a tulajdonságok sosem üresek.

Az `ASPError.Number` tulajdonság mindig visszaad egy értéket. Ha semmilyen hiba sem keletkezett és a `Server.GetLastError` metódussal létrehozunk egy `ASPError` objektumot, akkor az objektum `Number` tulajdonságának értéke `0`. Ha viszont előfordult egy hiba, akkor `Number` a szóban forgó hiba kódját (mely nem azonos az `ASPCode` által visszaadott kóddal) tartalmazza. Ha egy szkript hibáról van szó, akkor a hiba kódja hexadacimális alakban egyenlő `800A0000 + script_error_code`, ahol *`script_error_code`* a szkriptértelmező által visszaadott hibakód.

Az `ASPError6_vbs.asp` példaállománnyal előidézett hiba (osztás nullával) szkriptkódja 11. Ezek szerint a `Number` tulajdonság a `800A000B` hexadecimális, illetve `-2146828277` decimális értéket tartalmazza. Az `ASPError7_vbs.asp` példaállományban előfordult hiba `800A0409` hexadecimális, illetve `-2146827255` decimális értékkel tölti fel az `ASPError.Number` tulajdonságot, mivelhogy a hiba (le nem zárt karakterlánc) szkriptkódja `1033`.

A `Line` és `Column` tulajdonságok a hibát kiváltó állományon belül a hiba helyét, sort, illetve oszlopot adják meg – ha ez az információ a hiba bekövetkezésekor elérhető.

Futásidőben előforduló hibák esetében a `Column` tulajdonság értéke legtöbbször `-1`, jelezve, hogy a pontos hely nem állapítható meg. Fordítási hibák esetén a `Line` és `Column` tulajdonságok általában pontosan utalnak a hiba helyére.

Végül nézzük meg a `Source` tulajdonságot. Ha egy fordítási hibáról van szó, akkor ez a tulajdonság a forráskód hibás sorát tartalmazza. Ebben az esetben a `Line` és a `Column` tulajdonságok is értelmezhető értékeket tartalmaznak. Futási hibák esetében ez a tulajdonság egy üres karakterláncot tartalmaz.

2.8.2 Egyéni hibajelentő lapok elkészítése

Az előbbi példalapok megtekintésénél, a kedves Olvasó bizonyára észrevette, hogy hibajelentő lapjaink eltérnek az IIS eredeti hibajelentő lapjaitól, más szóval egyéni hibajelentő lapot használtam. Ebben a szakaszban ilyen egyéni hibajelentő lapok előállításával foglalkozunk.

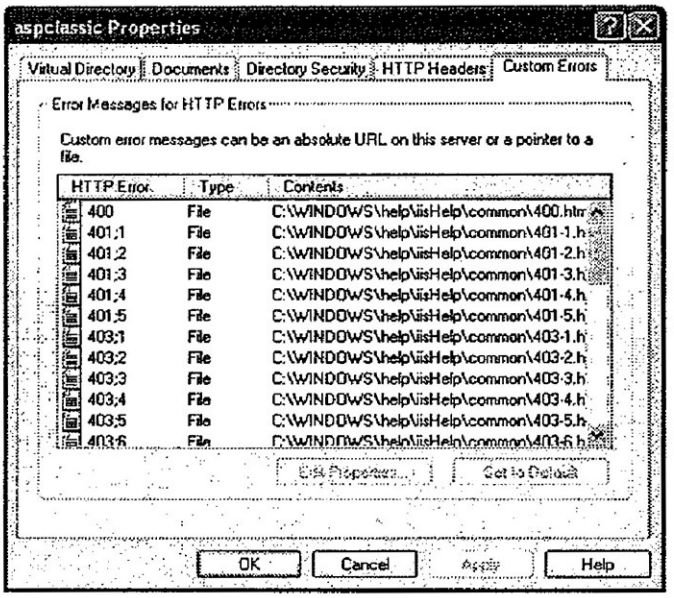
Egyéni hibajelentő lapok használata már csak azért is indokolt, mert az eredeti IIS lapok bizonyos esetekben kódrészletet is megjelenítenek, például egy adatbázis elérési útját, ami egy „élesre töltött" alkalmazás esetében nem kívánatos.

Amint láttuk az `ASPError` objektum tulajdonságai egy részletes leírást adnak a bekövetkezett hibáról. Következésképpen elegendő ezeket a tulajdonságokat lekérdeznünk és megfelelően megjelenítenünk. Itt gondoskodhatunk arról, hogy a kódrészleteket tartalmazó jelentéseket ne írjuk ki, illetve más jelentéssel helyettesítsük.

Mielőtt egy egyéni hibajelentő lap előállításával foglalkoznánk, tisztáznunk kell még egy kérdést: hogyan helyettesítjük az IIS eredeti hibajelentő lapját a sajátunkkal?

Ha egy ASP lapon egy hiba keletkezett és emiatt IIS nem tudja a kliens böngészőjéhez továbbküldeni, akkor a hibás lap helyett a szerver egy előre elkészített ASP vagy HTML lapot küld az ügyfélgépnek. Ezek az eredeti hibajelentő lapok a `WINNT\Help\iishelp\common\` mappában találhatók és természetesen egy böngészővel megtekinthetők, illetve egy lapszerkesztővel a forráskódjukat is megnézhetjük. Egy hiba keletkezésénél az IIS megállapítja a hiba kódját és a hibakódtól függően ezekből a hibajelentő lapokból küld vissza egyet a lekérdező gépnek. A hibakódok és a hibajelentő lapok megfeleltetése egy listában van meghatározva. Ezt a listát megtekinthetjük és az egyes hibakódokhoz egyéni hibajelentő lapokat feleltethetünk meg.

Nyissuk ki az MMC konzolt és kattintsunk az egér jobb gombjával az `aspclassic` virtuális mappára! A megjelenő menüben válasszuk ki a **Properties** menüpontot, majd a megjelenő párbeszédablakban válasszuk ki a **Custom Error** lapot. A megjelenő listában a hibakódokat és a megfelelő hibajelentő lapokat láthatjuk (2.32. ábra).

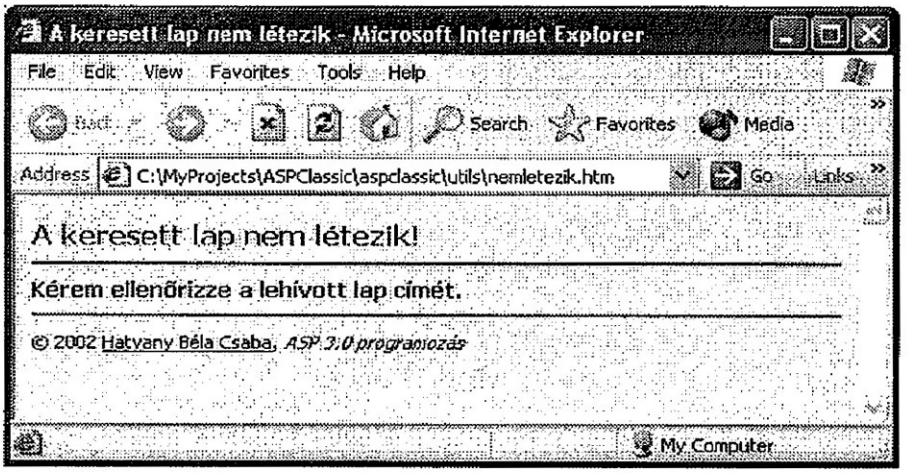


2.32. ábra Az IIS hibajelentő lapjai

A lehetséges hibák egy hibakóddal, bizonyos esetekben egy további alkóddal vannak felsorolva. Minden hibakódnak egy állomány vagy egy lap URL-je felel meg, amit a szerver a hiba bekövetkezése esetén a kliensnek küld. Például, a `404` kódnak a `C:\WINNT\help\iisHelp\common\404.htm` állomány felel meg a híres **The page cannot be found** (a lap nem létezik) jelentéssel. Az ASP hibák az `500`-as kódszám alatt vannak összegyűjtve. Szintaxis hiba esetén a kódszám `500;100`. Ennek a hibakódnak az `/iisHelp/common/500-100.asp` lap felel meg. Mint látjuk, egy hibakódot egy HTML állomány fizikális elérési útjával, vagy egy HTML vagy ASP lap virtuális elérési útjával párosíthatunk.

A következőkben a `404` és az `500;100` hibakódoknak fogunk egyéni hibajelentő lapokat megfeleltetni.

Írjunk egy egyszerű HTML lapot, amivel értesítjük látogatónkat, hogy az általa keresett lap nem létezik! Én a 2.33. ábrán látható lapot használtam.



2.33. ábra Egyéni hibajelentő lap a `400` hibakód részére

Mentsük el a lapot az alkalmazásunk egyik mappájába! A mi alkalmazásunk esetében, én a lapot `nemletezik.htm` név alatt az `/asplassic/utils` mappába helyeztem el. Mos kössük be a lapot az IIS listájába! Az előbbi párbeszédablak listájában kattintsunk a 404-es hibakódra, majd az **Edit Properies...** gombra. A megjelenő ablakon még egyszer ellenőrizhetjük a kiválasztott hibakódot (**Error Code: 404**), a hozzá tartozó alapértelmezett szöveget (**Default Text: Not Found**), a hibakódhoz rendelt üzenettípusát (**Message Type: File**) valamint a hibajelentő állományt (**File: C:\WINNT\help\iisHelp\common\404.htm**). Változtassuk meg az üzenet típusát URL-re és írjuk be a **URL** szövegmezőbe saját hibajelentő lapunk URL-jét: `/aspclassic/utils/nemletezik.htm`! (Az egyéni hibajelentő lapok URL-jét itt mindig a *domain*-név elhagyásával írjuk.). Kattintsunk az **OK** gombra és a **Properties** ablak **Apply** gombjára! Ezzel be is fejeztük a `404`-es hiba hozzárendelését egy egyéni hibajelentő laphoz.

Próbáljuk máris ki! Indítsuk el a böngészőnket és hívjuk le a `http://localhost/aspclassic/biztosnincsitt.htm` lapot, amiről biztosan tudjuk, hogy nem létezik! Ha mindent helyesen csináltunk, akkor a 2.33. ábrán látható lapnak kell megjelenni a böngészőnkben. Ha egy másik ASP alkalmazásból hívunk le egy nem létező lapot, akkor ismét az IIS hibajelentő lapja értesít a hibáról, mivelhogy az előbb saját lapunkat csak az `aspclassic` alkalmazáshoz kötöttük.

Befejezésül nézzük meg az `aspclassic` alkalmazásban használt hibajelentő lapot az `500;100` hibakódot viselő hiba esetében. A hibajelentő lapot `CustomErrorPage_vbs.asp` név alatt az `/aspclassic/utils` mappába helyeztem el és ezt a lapot kapcsoltam az `500;100` hibakódhoz. Ugyanezt a lapot, lényegtelen módosításokkal a jelenlegi szakasznak megfelelő `ASPError` mappába is megtaláljuk és így könnyen letölthetjük és forráskódját is megtekinthetjük a böngészőben. A `CustomErrorPage_vbs.asp` ASP lap a VBScript szkriptnyelvet alkalmazza, ez viszont nem akadálya annak, hogy egy JScript szkriptnyelvet használó lap hibáiról is megfelelő információt közöljön: a kommunikáció a hibajelentő lap és az IIS között nem függ a szkriptnyelvtől.

Az `ASPError` objektum bemutatásával befejeztük az ASP belső objektumainak tárgyalását. Az ASP belső objektumain kívül egy egész sor más objektummal is bővíthetjük az ASP alkalmazásokat. A következő fejezetben VBScript Scripting Runtime Library függvénykönyvtár objektumait fogjuk megismerni.

3. A Scripting objektumok

A második fejezetben tárgyalt objektumok az ASP belső objektumaiként ismeretesek. A belső objektumok mindig rendelkezésünkre állnak, nem kell külön létrehozni. A belső objektumok mellett az ASP olyan külső objektumokat is felhasználhat, melyek nem tartoznak a szűkebb értelemben vett ASP-hez. Ezek a külső, telepíthető objektumok rendkívül kibővíthetik az ASP-vel megoldható feladatok skáláját. Ilyen külső objektumok közé tartoznak a *Scripting Runtime Library* könyvtár által telepített objektumok is. A *Scripting Runtime Library* könyvtár a `scrrun.dll` DLL formájában az IIS telepítésével együtt, pontosabban a VBScript és JScript szkriptnyelveket értelmező Active Scripting gép telepítésével együtt kerül a gépünkre. Ezért, a *Scripting Runtime Library* könyvtár által definiált objektumokat röviden `Scripting` objektumoknak is szoktuk nevezni.

A `Scripting` objektumok hierarchiájában a következő három objektum központi szerepet játszik:

- a `FileSystemObject` objektum lehetővé teszi, hogy a szerver állományrendszerét az ASP lapokból elérhessük;
- a `TextStream` objektum lehetővé teszi, hogy a szerveren létező állományokat elérhessük, hogy új állományokat hozhassunk létre vagy meglévő állományokat másolhassunk, máshova áthelyezhessünk vagy törölhessünk,
- a `Dictionary` objektum segítségével, a hagyományos tömbökhöz hasonlóképpen, értékeket tárolhatunk, azzal a különbséggel, hogy a `Dictionary` objektum tagjait indexek helyett kulcsszavakkal láthatjuk el.

A `FileSystemObject` objektum a következő három objektumot, illetve gyűjteményt bocsátja rendelkezésünkre:

- a `Drive` objektumot és a `Drives` gyűjteményt;
- a `Folder` objektumot és a `Folders` gyűjteményt;
- a `File` objektumot és a `Files` gyűjteményt.

A `FileSystemObject` objektum segítségével, az alárendelt objektumok és gyűjtemények, valamint a `TextStream` objektum segítségével a szerver állományrendszerét teljes ellenőrzés alá vehetjük.

Ebben a fejezetben a `Scripting` objektumokat fogjuk megismerni.

3.1 A FileSystemObject (FSO) objektum

A FileSystemObject (FSO) objektum a Scripting objektumok központjában áll. Segítségével alkalmazásunk elérheti az alárendelt objektumokat vagy gyűjteményeket. Ugyanis azért, hogy bármelyiket az alárendelt objektumok vagy gyűjtemények közül használhassunk, először egy FileSystemObject objektumot kell létrehoznunk.

Egy FSO objektumot a Server objektum CreateObject metódusával:

```
Dim objFSO
Set objFSO = _
Server.CreateObject("Scripting.FileSystemObject")
```

vagy az <OBJECT> elem felhasználásával:

```
<OBJECT RUNAT="SERVER" SCOPE="PAGE" ID="objFSO"
PROGID="Scripting.FileSystemObject">
```

hozhatunk létre. Miután az FSO objektumot létrehoztuk, felhasználhatjuk bármelyik metódusát és tulajdonságát. Az így létrehozott FSO objektum csak az állományon belül létezik. Miután már nincs szükségünk rá, tanácsos eltávolítani a szerver memóriájából:

```
Set objFSO = Nothing
```

A scrrun.dll könyvtár az említett objektumok mellett sok konstanst is definiál. Ezek a Scripting konstansok, melyek a Scripting objektumok metódusainál vagy tulajdonságainál fontos szerepet játszanak, amint ezt nemsokára látni fogjuk. A Scripting konstansokat többféleképpen definiálhatjuk egy ASP lapban. Egyik lehetőségünk abban áll, hogy numerikus értékükkel adjuk meg abban az ASP lapban, amelyikben használjuk:

```
Const ForReading = 1
```

Egy másik eljárás a scrrun.dll könyvtár típuskönyvtárát (*type librabry*) metaadatként (*metadata*) fűzi ASP lapunkhoz:

```
<!--METADATA TYPE="typelib" FILE="C:\WINNT\SYSTEM32\scrrun.dll" -
->
```

A FILE paraméter által megadott elérési utat szükség esetén aktualizálni kell.

A harmadik lehetőséget az #include rendelkezés (*directive*) adja. Az #include rendelkezéssel egy létező állományt beágyazhatunk (*include*) egy ASP lapba. Az #include rendelkezés szintaxisa a következő:

```
<!-- #include PathType = FileName -->
```

ahol *PathType* `file` vagy `virtual` lehet és *FileName* a beágyazandó állomány (`file` típus esetében fizikai, `virtual` típus esetében virtuális) elérési útja. A beágyazandó állomány bármilyen nevet és kiterjesztést viselhet. Leggyakrabban a beágyazott állományok `.inc` kiterjesztésűek. A *FileName* elérési úttal megadott állomány azon a helyen lesz a rendelkezést tartalmazó állományba beágyazva, ahol a rendelkezés található. Más szóval, az ASP lapot értelmező gép áttér a *FileName* elérési út által megadott állomány értelmezésére, mihelyt az `#include` rendelkezést értelmezte és ezután nyomban visszatér az eredeti lap értelmezésére.

Ezek szerint egyes, a `Scripting` konstansokat a következő `scripting.inc` állománnyal ágyazhatunk be szükség esetén:

```
<%
' scripting.inc - Scripting Constants
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Members of Scripting.CompareMethod
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const BinaryCompare = 0
Const TextCompare = 1
Const DatabaseCompare = 2

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Members of Scripting.DriveTypeConst
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const UnknownType = 0
Const Removable = 1
Const Fixed = 2
Const Remote = 3
Const CDRom = 4
Const RamDisk = 5

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Members of Scripting.FileAttribute
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const Normal= 0
Const ReadOnly = 1
Const Hidden = 2
Const System = 4
Const Volume = 8
Const Directory = &H10 ' 16
Const Archive = &h20 ' 32
Const Alias = &H400 ' 1024
Const Compressed = &H800 ' 2048

'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Members of Scripting.IOMode
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8
```

```
'''''''
```

A `GetDrive`, `GetFolder` és `GetFile` metódusok egy `Drive`, `Folder`, illetve `File` objektumot hoznak létre. Ezeket az objektumokat a fejezet következő szakaszaiban fogjuk megismerni és ott fogjuk a `FileSystemObject` objektum `GetDrive`, `GetFolder` és `GetFile` metódusainak alkalmazását is látni. Egyenlőre jegyezzük meg, hogy mindhárom metódusnak egy karakterlánc típusú paraméterrel kell a megfelelő specifikációt átadnunk:

```
Dim objFSO
Dim objDrive
Dim objFolder
Dim objFile

Set objFSO = _
  Server.CreateObject("Scripting.FileSystemObject")
Set objDrive = objFSO.GetDrive("C:")
Set objFolder = objFSO.GetFolder("C:\myFolder")
Set objFile = objFSO.GetFile("C:\myFolder\myFile")
```

A `GetDrive`, `GetFolder` és `GetFile` metódusokkal csak akkor hozhatunk létre egy `Drive`, `Folder`, illetve `File` objektumot, ha a metódusnak továbbadott specifikáció egy fizikálisan létező meghajtót, mappát, illetve állományt jelöl meg. Ellenkező esetben a metódusok egy hibát generálnak.

A `CopyFolder`, illetve `CopyFile` metódussal egy vagy több mappát vagy állományt másolhatunk egy megadott célhelyre. A metódusok szintaxisa a következő:

```
objFSO.CopyFolder(source, destination [, overwrite])
objFSO.CopyFile(source, destination [, overwrite])
```

ahol

- *source* a `CopyFolder` metódus esetében egy mappa- vagy állomány-specifikáció, a `CopyFile` metódus esetében egy állomány-specifikáció. Mindkét esetben *source* karakterhelytartókat (*wildcard*) tartalmazhat;
- *destination* egy karakterlánc, mely a célhelyet adja meg;
- *overwrite* értéke **`True`** (alapértelmezett érték) vagy **`False`** lehet. **`True`** érték esetében állományok és mappák felül lesznek írva.

Ha *source* karakterhelytartókat tartalmaz vagy *destination* utolsó karaktere `"\"`, akkor *destination* egy létező vagy egy létrehozandó mappát ad meg. Mindkét esetben a következő szabályok alkalmazhatók:

- ha a *destination* által megadott mappa nem létezik: az `FSO` objektum létrehozza a szükséges mappát és a *source* által megadott mappákat és/vagy állományokat átmásolja az újonnan létrehozott mappába. Ez a leggyakrabban előforduló eset;

- ha a *destination* által megadott célhely egy létező állománynak felel meg: az FSO objektum egy hibát generál;
- ha a *destination* által megadott célhely egy létező mappának felel meg: az FSO objektum megkísérli a *source* által megadott mappákat és/vagy állományokat a célmappába másolni. Ha a célmappa a *source* által megadott azonos nevű mappákat és/vagy állományokat tartalmaz, és ha *overwrite* értéke **False**, akkor az FSO objektum egy hibát generál;
- ha a *destination* által megadott célhely egy csak olvasható (*read-only*) mappa, akkor az FSO objektum egy hibát generál.

A MoveFolder, illetve MoveFile metódussal egy vagy több mappát vagy állományt áthelyezhetünk egy megadott célhelyre. A metódusok szintaxisa a következő:

```
objFSO.MoveFolder(source, destination )
objFSO.MoveFile(source, destination )
```

ahol

- *source* a MoveFolder metódus esetében egy mappa- vagy állomány-specifikáció, a MoveFile metódus esetében egy állomány-specifikáció. Mindkét esetben *source* karakterhelytartókat (*wildcard*) tartalmazhat, de csak az elérési út utolsó komponensében;
- *destination* egy karakterlánc, mely a célhelyet adja meg.

Ha *source* utolsó komponense karakterhelytartókat tartalmaz vagy *destination* utolsó karaktere "\", akkor *destination* egy létező vagy egy létrehozandó mappát ad meg. Mindkét esetben a következő szabályok alkalmazhatók:

- ha a *destination* által megadott mappa nem létezik: az FSO objektum létrehozza a szükséges mappát és a *source* által megadott mappákat és/vagy állományokat áthelyezi az újonnan létrehozott mappába. Ez a leggyakrabban előforduló eset;
- ha a *destination* által megadott célhely egy létező állománynak vagy mappának felel meg: az FSO objektum egy hibát generál.

A DeleteFolder és DeleteFile metódusokkal mappákat és állományokat törölhetünk. A metódusok szintaxisa:

```
objFSO.DeleteFolder (spec[, force] )
objFSO.DeleteFile (spec[, force] )
```

ahol

- *spec* a `DeletFolder` metódus esetében egy mappa- vagy állomány-specifikáció, a `DeletFile` metódus esetében egy állomány-specifikáció. Mindkét esetben *spec* karakterhelytartókat (*wildcard*) tartalmazhat, de csak az elérési út utolsó komponensében;
- *force* értéke **True** vagy **False** (alapértelmezett érték) lehet. **True** érték esetében a csak olvasható (*read-only*) állományok és mappák törölve lesznek.

Ha az `FSO` objektum nem találja meg a törlendő mappát vagy állományt, akkor egy hibát generál.

A `CreateFolder` metódussal egy mappát hozhatunk létre:

```
Dim objFSO
Dim objFolder
Set objFSO = _
Server.CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder(strFolder)
```

Ebben a kódrészletben *strFolder* a létrehozandó mappa elérési útja. A metódus visszatérési értéke egy `Folder` objektum.

A `CreateTextFile` metódussal egy állományt hozhatunk létre. A metódus szintaxisa:

```
objFSO.CreateTextFile(filename[, overwrite[, unicode]])
```

ahol

- *filename* a létrehozandó állomány specifikációja;
- *overwrite* értéke **True** vagy **False** (alapértelmezett érték) lehet. Ha a paraméter értéke **True**, akkor meglévő állományok felül lesznek írva. Ellenkező esetben meglévő állományok nem lesznek felülírva és az FSO objektum egy hibát generál.
- *unicode* értéke **True** vagy **False** (alapértelmezett érték) lehet. Ha a paraméter értéke **True**, akkor az új állomány Unicode állományként lesz létrehozva. Ellenkező esetben az új állomány ASCII állományként lesz létrehozva.

A következő kódrészlet megmutatja, miként hozhatunk létre egy állományt a `CreateTextFile` metódus segítségével:

```
Dim objFSO
Dim objTextStream
```

```
Set objFSO = _
Server.CreateObject("Scripting.FileSystemObject")
Set objTextStream = objFSO.CreateTextFile ("C:\myFile")
```

A metódus visszatérő értéke egy `TextStream` objektum, amivel egy külön szakaszban foglalkozunk.

Az `OpenTextFile` metódus olvasásra vagy írásra megnyit egy létező vagy nem létező és ebben az esetben létrehozandó szövegállományt. Felépítése a következő:

```
objFSO.OpenTextFile(filename[, iomode[, create[, format]]])
```

ahol

- *filename* - megnyitandó állomány abszolút vagy relatív elérési útja;
- *iomode* meghatározza a bemenet/kimenet módját. Lehetséges értékei:
  - 1 - megfelel a `ForReading Scripting` konstansnak: az `FSO` objektum az állományt olvasásra nyitja meg;
  - 2 - megfelel a `ForWriting Scripting` konstansnak: az `FSO` objektum az állományt írásra nyitja meg;
  - 8 - megfelel a `ForAppending Scripting` konstansnak: az `FSO` objektum az állományt szöveghozzáfűzésre nyitja meg;
- *create* értéke **True** vagy **False** (alapértelmezett érték) lehet. Ha a paraméter értéke **True** és *filename* egy nem létező állomány, akkor az `FSO` objektum megnyitása előtt létrehozza az állományt;
- *format* meghatározza a megnyitandó állomány formátumát. Lehetséges értékei:
  - 0 - megfelel a `TristateFalse Scripting` konstansnak: az `FSO` objektum az állományt ASCII formátumban nyitja meg. Ez az alapértelmezett érték;
  - -1 - megfelel a `TristateTrue Scripting` konstansnak: az `FSO` objektum az állományt Unicode formátumban nyitja meg;
  - -2 - megfelel `TristateUseDefault Scripting` konstansnak: az `FSO` objektum az állományt a rendszer alapértelmezett formátumában nyitja meg.

Ennek a metódusnak a visszatérő értéke szintén egy `TextStream` objektum. A következő kódrészlet egy létező vagy létrehozandó szövegállomány megnyitását illusztrálja

```
Dim objFSO
Dim objTextFile
Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
Set objTextStream = objFSO.OpenTextFile ("C:\myFile", True)
```

A `GetSpecialFolder` metódussal az operációs rendszer különleges mappáit érhetjük el. Visszatérési értéke egy `Folder` objektum:

```
objFolder = objFSO.GetSpecialFolder(specialfolderconst)
```

A metódus paramétere egy konstans, mely a következő értékeket veheti fel:

```
WindowsFolder = 0
SystemFolder = 1
TemporaryFolder = 2
```

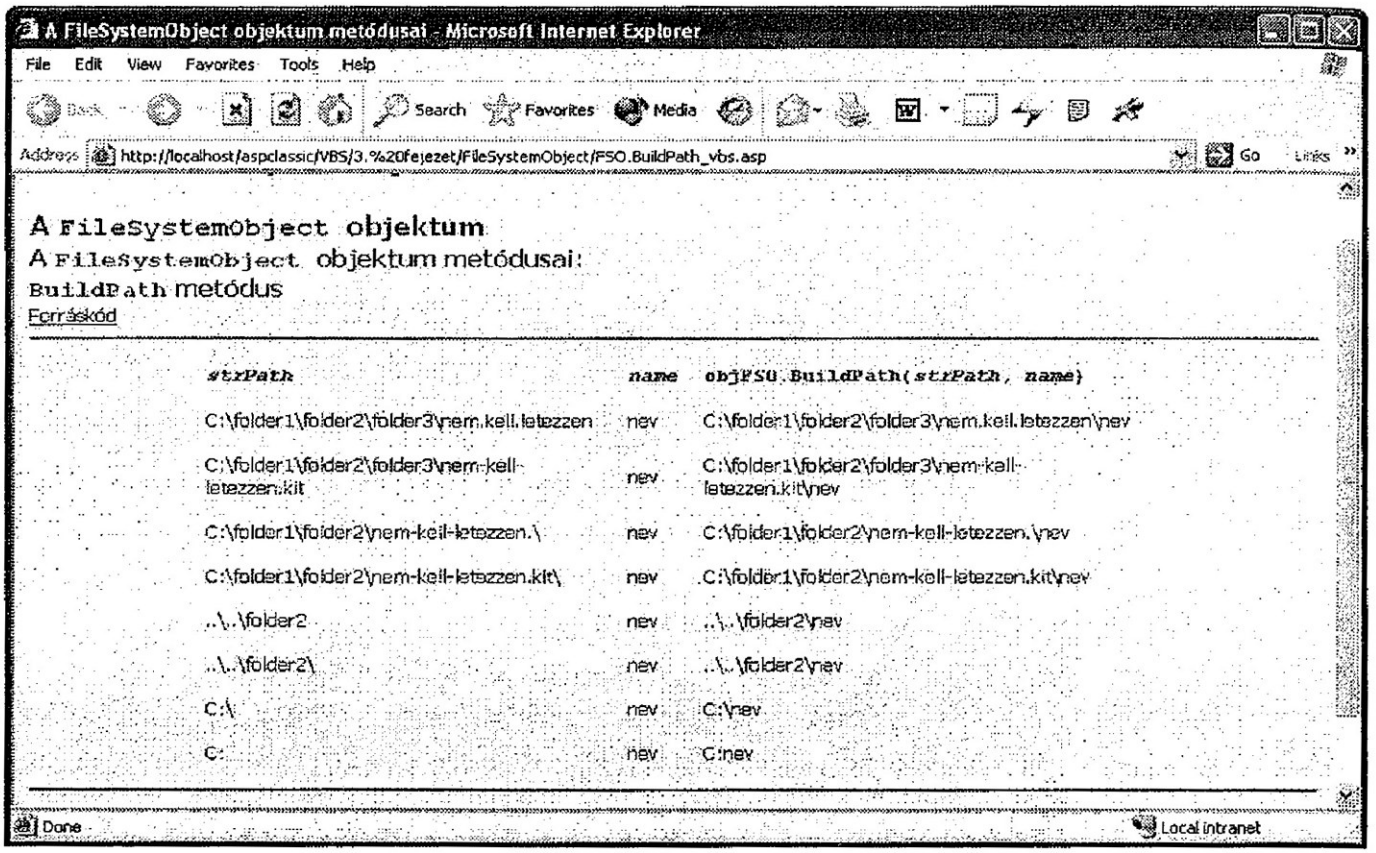
A konstansok a Windows-mappát (alapértelmezésben `C:\WinNT`), a rendszermappát (alapértelmezésben `C:\WinNT\System32`) és az ideiglenes mappát (alapértelmezésben `C:\WinNT\Temp`) azonosítják. A `GetSpecialFolder` metódus alkalmazását a `FSO.GetSpecialFolder_vbs.asp` lap mutatja be.

A `FileSystemObject` objektum következő metódusai nem foglalkoznak a `FileSystemObject` objektumnak alárendelt objektumaival, hanem karakterláncokkal különböző segédműveleteket végeznek el. Visszatérési értékük mindig egy karakterlánc és nem egy objektum. Elnevezésükben (egy kivételével) a `Name` rész pontosan erre a sajátosságukra utal.

A `BuildPath` metódus két paramétert vár:

```
FileSystemObject.BuildPath(strPath, name)
```

A *strPath* paraméter egy elérési utat ad meg, a *name* paraméter egy állományt vagy mappát nevez meg. A *strPath* paraméter által megadott elérési út egy szabályosan felépített abszolút vagy relatív elérési út kell legyen, azonban nem kell feltétlenül egy létező mappát meghatározzon. Például `"C:"` nem egy szabályos elérési út, hiszen nem létezhet egy mappa sem ezzel az elérési úttal. Ezzel szemben `"C:\"` a `C` meghajtó törzsmappájának szabályos elérési útja. A *name* paraméter egy állomány vagy egy mappa szabályos nevének kell lenni. Ha ezek a feltételek fennállnak, akkor a `BuildPath` metódus a *strPath* és a *name* paraméterből egy szabályos elérési utat szerkeszt, szükség szerint beillesztve egy `"\"` elválasztó karaktert a két paraméter közé. A metódus nem hoz létre sem egy mappát sem egy állományt. A `BuildPath` metódus alkalmazását a `3. fejezet/FileSystemObject` mappában elhelyezett `FSO.BuildPath_vbs.asp` példalap mutatja be (3.1. ábra).



3.1. ábra A `BuilPath` *metódus alkalmazása*

A `GetDriveName` metódus egy *drivespec* meghajtó-specifikációból kiolvassa a meghajtó betűnevét:

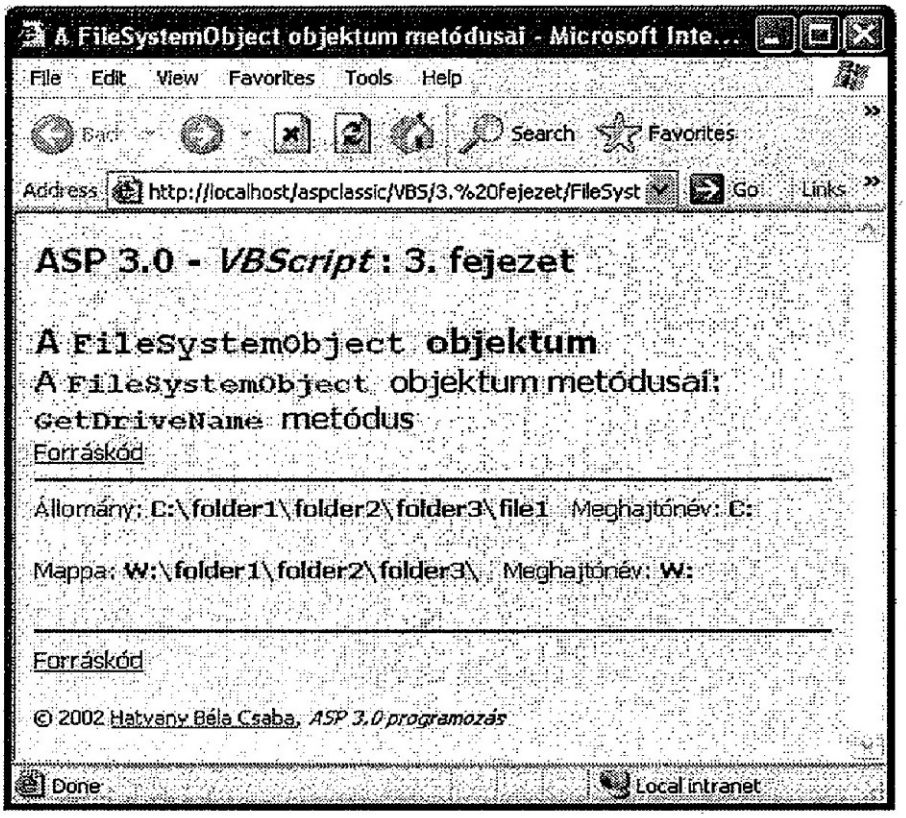
```
strDriveLetter = objFSO.GetDriveName(drivespec)
```

A metódus alkalmazását a `FSO.GetDriveName_vbs.asp` lap mutatja be és a 3.2. ábrán láthatjuk.

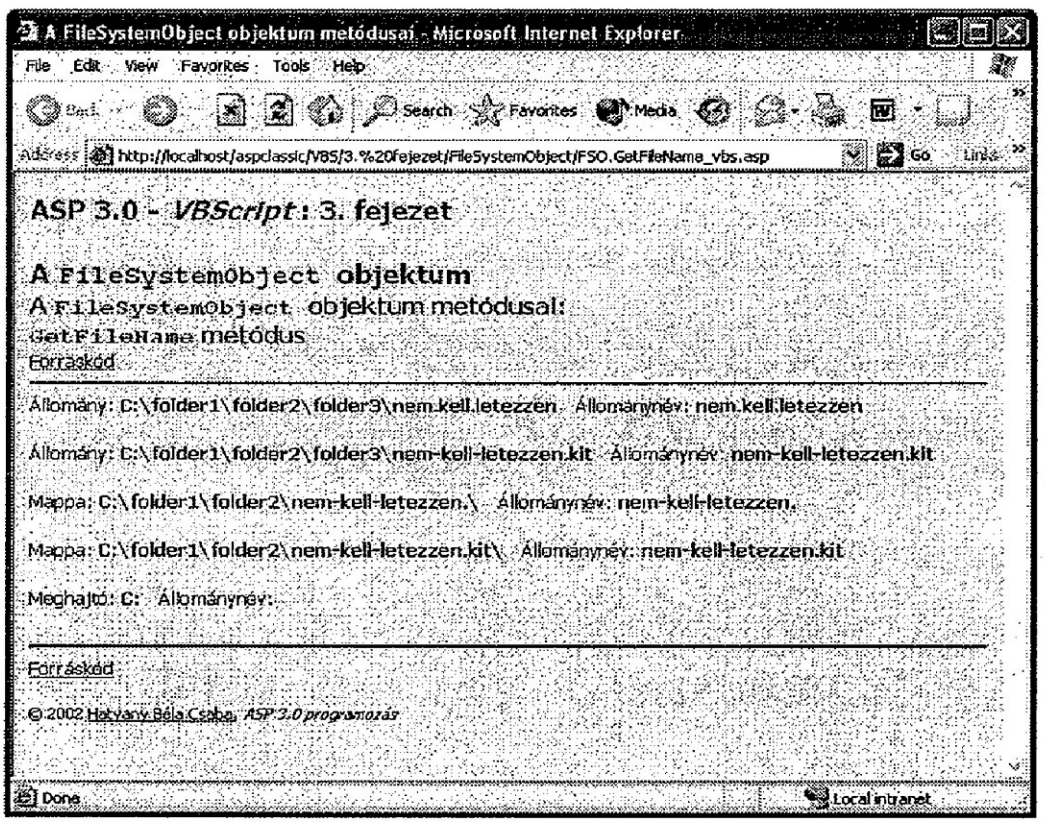
A `GetFileName` metódus egy *pathspec* állomány- vagy mappa-specifikáció utolsó komponensét adja vissza:

```
strFileName = objFSO.GetFileName (pathspec)
```

Alkalmazását az `FSO.GetFileName_vbs.asp` lap, illetve a 3.3. ábra mutatja be.



3.2. ábra A `GetDriveName` *metódus alkalmazása*

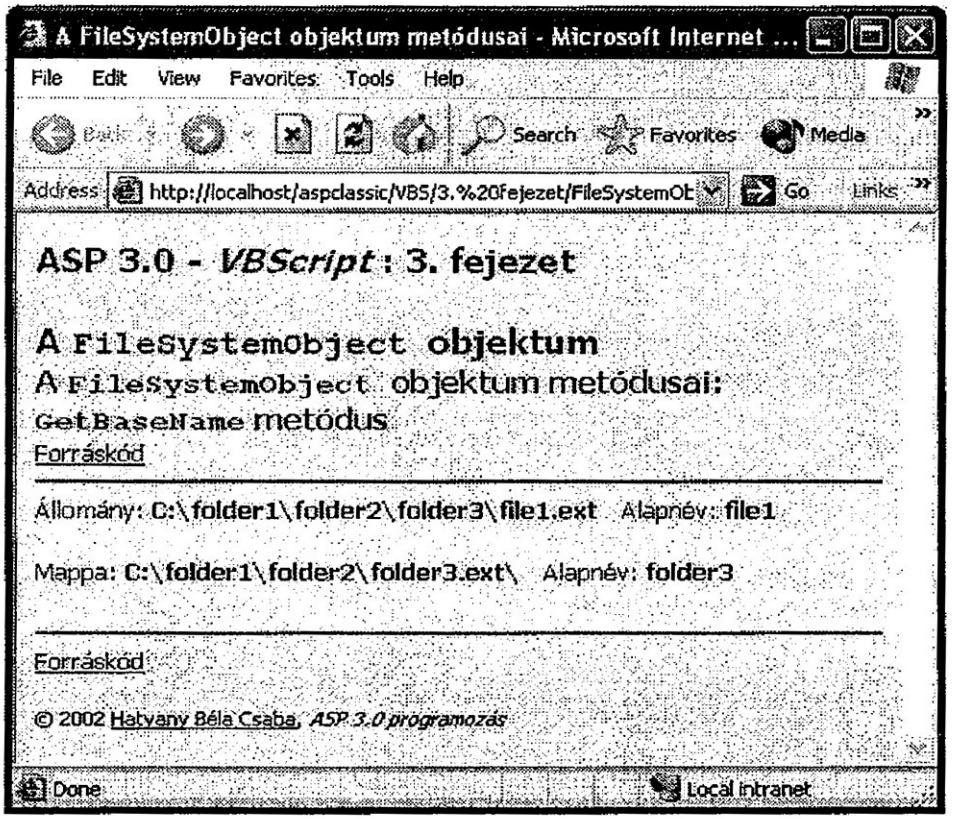


3.3. ábra A `GetFileName` *metódus alkalmazása*

A `GetBaseName` metódus egy *pathspec* állomány- vagy mappa-specifikáció alapnevét adja vissza:

```
strBaseName = objFSO.GetFileName (pathspec)
```

Emlékeztetőül, egy állomány- vagy mappa-specifikáció alapneve a specifikáció utolsó komponensének neve az esetleges kiterjesztés nélkül. Alkalmazását a `FSO.GetBaseName_vbs.asp` szemlélteti és a 3.4. ábrán láthatjuk.

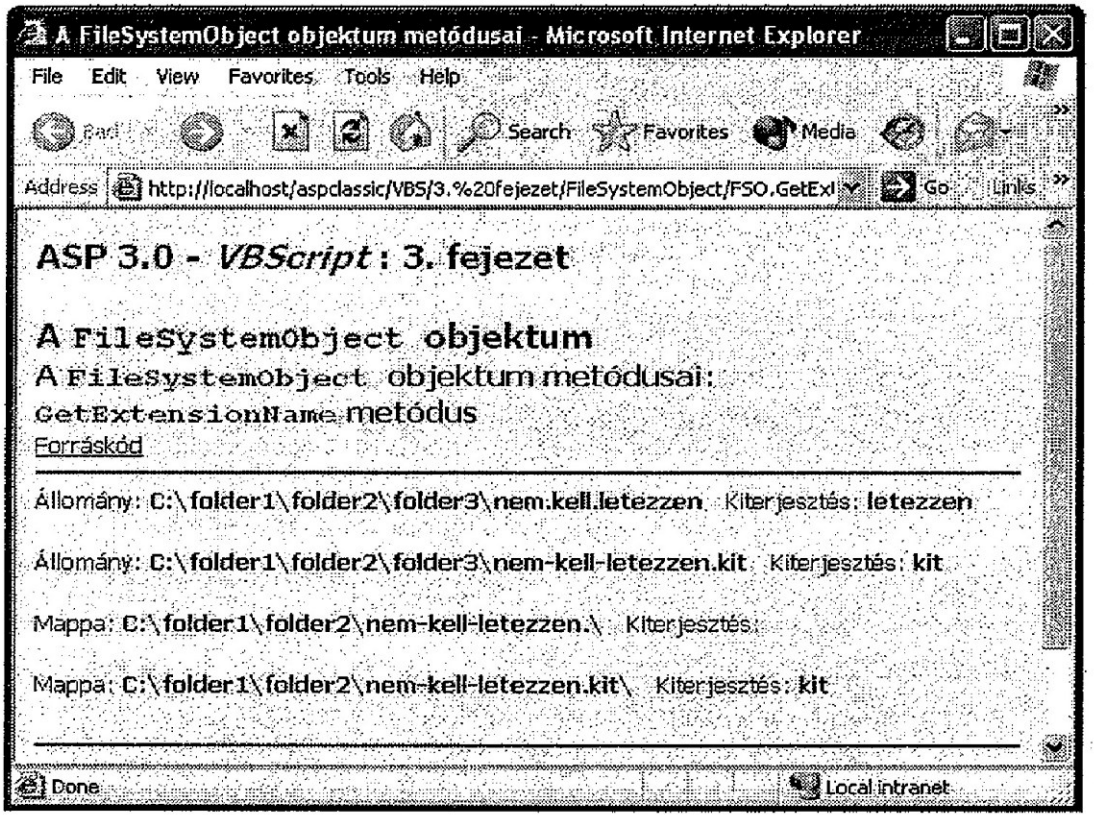


3.4. ábra A `GetBaseName` *metódus alkalmazása*

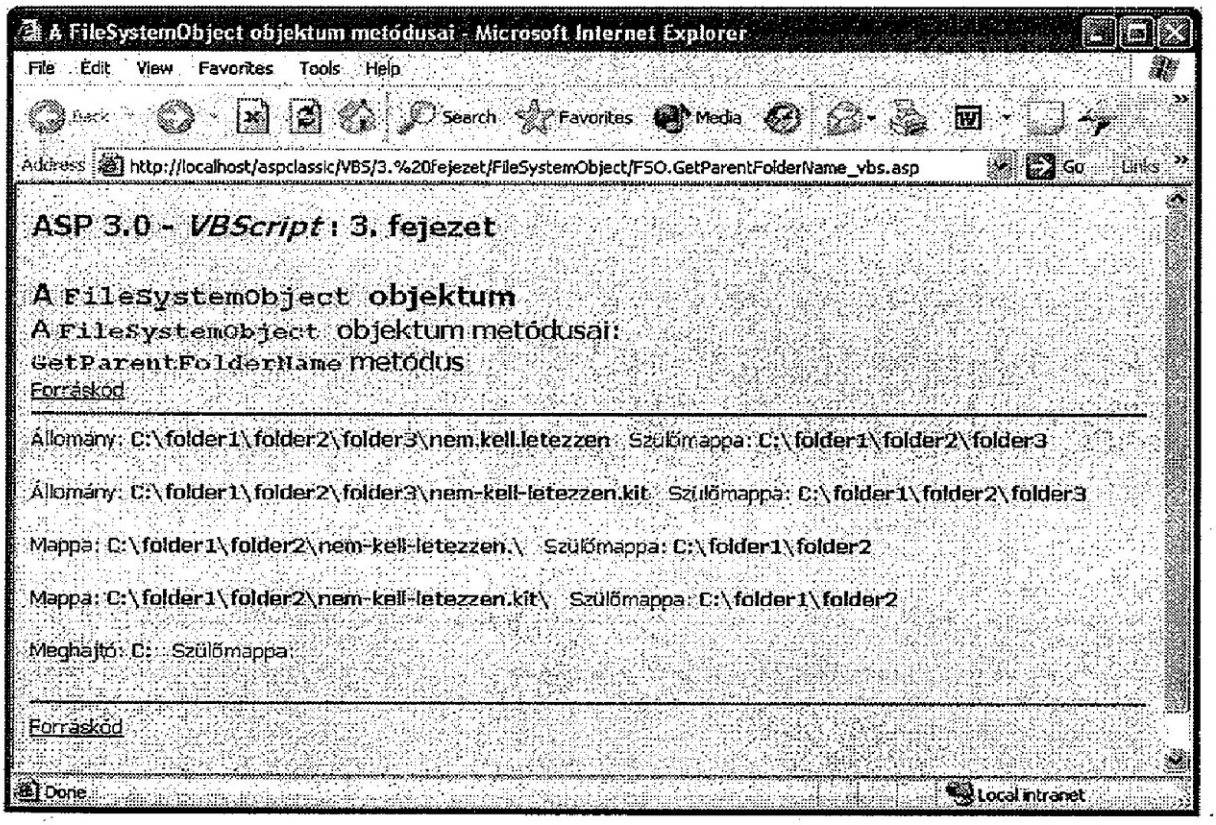
Ezzel a metódussal szemben a `GetExtensionName` metódus egy *pathspec* állomány- vagy mappa-specifikáció kiterjesztését adja vissza:

```
strExt = objFSO.GetExtensionName (pathspec)
```

Az `FSO.GetExtensionName_vbs.asp` lap bemutatja a metódus használatát, amit 3.5. ábrán is láthatunk.



3.5. ábra A `GetExtensionName` *metódus alkalmazása*



3.6. ábra A `GetParentFolderName` *metódus alkalmazása*

A `GetParentFolderName` metódus egy *pathspec* állomány- vagy mappa-specifikáció szülőmappáját, más szóval a specifikáció utolsó előtti komponensét adja vissza:

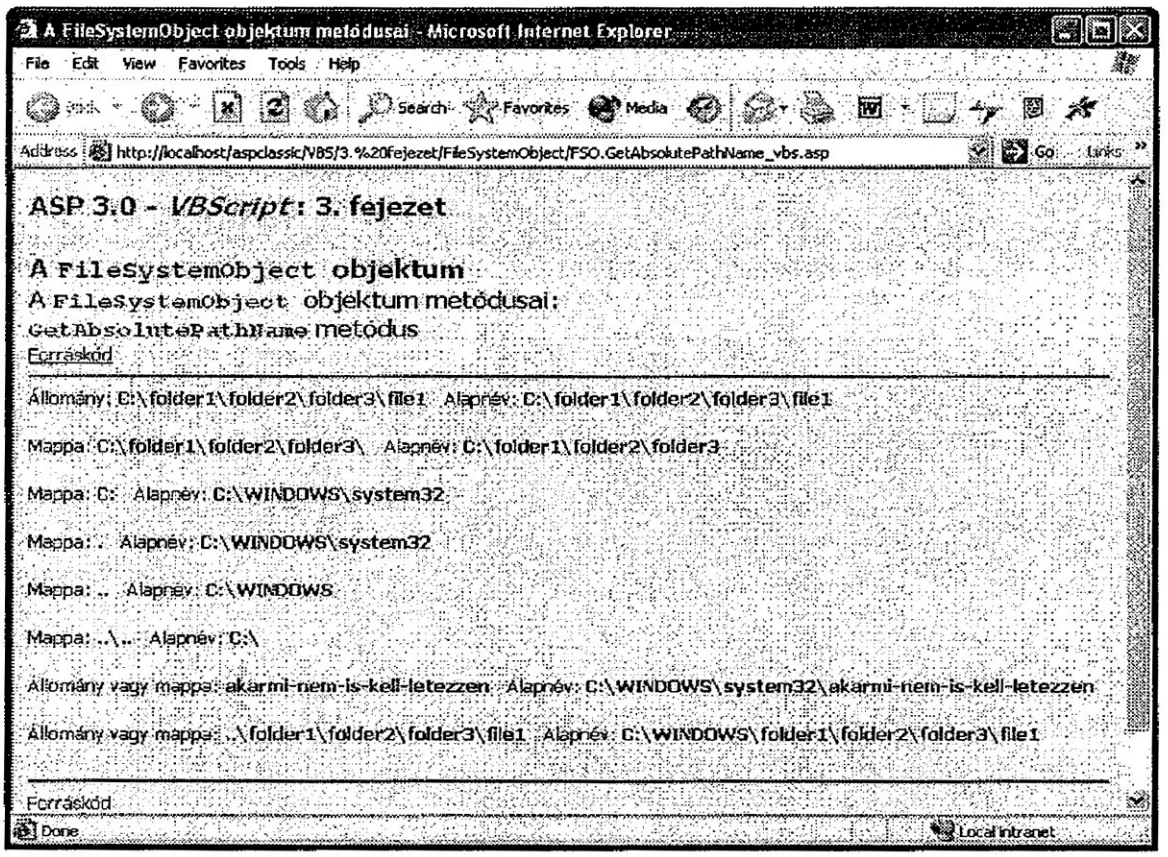
```
strParentFolder = objFSO.GetParentFolderName (pathspec)
```

Az `FSO.GetParentFolderName_vbs.asp` lap többször hívja meg ezt a metódust (3.6. ábra).

A `GetAbsolutePathName` metódus az aktuális mappához viszonyított *pathspec* relatív elérési utat abszolút formájára alakítja át és adja vissza:

```
strAbsolutePath = objFSO.GetAbsolutePathName (pathspec)
```

A 3.7. ábrán láthatjuk a `GetAbsolutePathName` metódus alkalmazását különböző relatív elérési utak esetében. Az ábra az `FSO.GetAbsolutePathName_vbs.asp` lapot mutatja a böngészőben.



3.7. ábra A `GetAbsolutePathName` *metódus alkalmazása*

A `FileSystemObject` objektum utolsó metódusa a `GetTempName` metódus, mely esetlegesen (*randomly*) generált állománynevet ad vissza. A metódus nem vár egy paramétert sem. A metódust az `FSO.GetTempName_vbs.asp` lapban alkalmaztam. Ha a lapot lehívjuk, majd a böngészőben ismételten frissítjük, akkor a lap esetlegesen generált állományneveket jelenít meg.

3.1.2 A `FileSystemObject` objektum `Drives` tulajdonsága

A `FisleSystemObject` objektum tárgyalásának elején megemlítettem, hogy az `FSO` objektumnak csak egy tulajdonsága van, a `Drives` tulajdonság. Most ezt a tulajdonságot fogjuk megvizsgálni.

A `FisleSystemObject` objektum `Drives` tulajdonsága csak olvasható. Értéke egy `Drive` objektumokból álló gyűjtemény:

```
colDrives = objFSO.Drives
```

A gyűjtemény `Drive` objektumai a szerver meghajtóinak felelnek meg. A `Drives` gyűjtemény a szokásos tulajdonságokkal rendelkezik: a `Count` tulajdonsággal, mely a gyűjtemény elemeinek számát adja vissza és az `Item` tulajdonsággal, mely segítségével hozzáférhetünk a gyűjteményben foglalt egyes meghajtókhoz. A `Count` tulajdonság alkalmazását az `FSO.Drives_vbs.asp` példalap mutatja be. Az `Item` tulajdonság alkalmazását a `Drive` objektummal foglalkozó szakaszban láthatjuk. Itt csak arra hívom fel a figyelmet, hogy az `Item` tulajdonság kulcsként egy meghajtóbetűt vár karakterlánc alakban:

```
objDrive = objFSO.Drives.Item("C")
```

Ha viszont a meghajtóbetűt egy változóba mentjük, akkor a következő kódrészlet hibát generálhat:

```
strDriveLetter="C"
objDrive = objFSO.Drives.Item(strDriveLetter)
```

Egy VBScript sajátosság, hogy az összes változó **`Variant`** típusú. A fenti kódrészletben ezért a `strDriveLetter` változót kifejezetten át kell alakítani karakterlánccá:

```
strDriveLetter="C"
objDrive = objFSO.Drives.Item(CStr(strDriveLetter))
```

A `FileSystemObject` objektum `Drives` tulajdonságával és `Drives` gyűjteményével a `Drive` objektum tárgyalásánál ismét találkozni fogunk.

3.2 A `Drive` objektum

A `Drive` objektum a `FileSystemObject` objektum alárendelt objektuma és a szerver lokális vagy hálózati meghajtóját képviseli. A `Drive` objektum csak a `FileSystemObject` objektumon keresztül érhető el. Ez azt jelenti, hogy egy `Drive` objektumot nem hozhatunk létre közvetlenül a `Server` objektum `CreateObject`

metódusával. Ez természetes is, ha figyelembe vesszük, hogy egy `Drive` objektum mindig a szerver egy létező meghajtóját képviseli.

Egy `Drive` objektumot a `FileSystemObject` objektum `Drives` tulajdonságával érhetünk el. Mint láttuk, a `Drives` tulajdonság egy `Drive` objektumokból álló gyűjteményt ad vissza. A rendelkezésre álló `Drive` objektumok a `Drives` gyűjtemény tagjai:

```
Dim objFSO
Dim colDrives
Dim objDriveC

Set objFSO = _
Server.CreateObject("Scripting.FileSystemObject")
Set colDrives = objFRO.Drives
Set objDriveC = colDrives.Item("C")
```

A `FileSystemObject` objektum `GetDrive` metódusáról tudjuk, hogy egy `Drive` objektumot ad vissza, ha paramétere egy fizikálisan létező meghajtó specifikációja:

```
Dim objFSO
Dim objDriveC
Set objFSO = _
Server.CreateObject("Scripting.FileSystemObject")
Set objDriveC = objFSO.GetDrive("C")
```

A `Drive` objektum nem rendelkezik metódusokkal, viszont a 12 tulajdonsága teljesen leír bármilyen meghajtót. A következő szakaszban a `Drive` objektum tulajdonságait fogjuk megismerni. Mint látni fogjuk, egy kivételével a `Drive` objektum tulajdonságai csak olvashatók (*read-only*), mivelhogy a meghajtók fizikális tulajdonságait adják meg.

3.2.1 A `Drive` objektum tulajdonságai

A `TotalSize`, `FreeSpace` és `AvailableSpace` tulajdonságok a meghajtó tárterületéről adnak információt.

A `TotalSize` tulajdonság a meghajtó össztárterületét adja meg, a `FreeSpace` tulajdonság a meghajtón levő szabad tárterületet, míg az `AvailableSpace` tulajdonság a meghajtón elérhető tárterületet adja meg bájtban kifejezve:

```
Dim lngTotalBytes
Dim lngFreeBytes
Dim lngAvailableBytes
```

```
lngTotalBytes = objDrive. TotalSize
lngFreeBytes = objDrive. FreeSpace
lngAvailableBytes = objDrive.AvailableSpace
```

Legtöbb esetben a szabad tárterület megegyezik ez elérhető tárterülettel. Elképzelhető azonban, hogy egyes felhasználók csak korlátozott mértékben használhatják egy meghajtó tárterületét. Ebben az esetben az `AvailableSpace` tulajdonság az érvényben levő korlátokat figyelembe véve adja meg az elérhető tárterület nagyságát.

A `DriveLetter` tulajdonság a meghajtóbetűt adja vissza:

```
Dim strDriveLetter
strDriveLetter = objDrive.DriveLetter
```

A `DriveType` tulajdonság egy egész számú értéket ad vissza, mely a meghajtó típusát adja meg:

```
Dim intDriveType
intDriveType = objDrive.DriveType
```

A `DriveType` tulajdonság által visszaadott értékek a következők lehetnek:

érték	meghajtó típus
0	ismeretlen
1	cserélhető meghajtó
2	nem cserélhető meghajtó
3	hálózati meghajtó
4	CDROM meghajtó
5	RAM meghajtó

Ha felhasználjuk a fejezet elején ismertetett `scripting.inc` állományt, akkor a következő függvénnyel megállapíthatjuk egy meghajtó típusát:

```
<%
 Function GetDriveType(strDriveSpec)
%>
<!-- #include file="scripting.inc" -->
<%
        Dim objFSO
        Dim objDrive

        Set objFSO = _
        Server.CreateObject("Scripting.FileSystemObject")
        Set objDrive = objFSO.GetDrive(strDriveSpec)
```

```
        Select Case objDrive.DriveType
                Case UnknownType
                        GetDriveType = "Ismeretlen"
                Case Removable
                        GetDriveType = "Cserélhetõ"
                Case Fixed
                        GetDriveType = "Nem cserélhetõ"
                Case Remote
                        GetDriveType = "Távoli"
                Case CDRom
                        GetDriveType = "CDROM"
                Case RamDisk
                        GetDriveType = "RAMDisk"
        End Select
        Set objDrive = Nothing
        Set objFSO = Nothing
  End Function
%>
```

Az `IsReady` tulajdonság visszaadott értéke **`True`**, ha a meghajtó készenlétben áll, illetve **`False`** ellenkező esetben:

```
Dim blnIsReady
blnIsReady = objDrive.IsReady
```

Például egy cserélhető meghajtó esetében, melyben nincs lemez, ez a tulajdonság **`False`** értéket ad vissza. Szintén **`False`** értéke van az `IsReady` tulajdonságnak távoli vagy lemez nélküli CDROM meghajtók esetében.

A `FileSystem` tulajdonság egy karakterláncot ad vissza, mely leírja a meghajtón használt állományrendszert:

```
Dim strFileSystem
strFileSystem = objDrive. StrFileSystem
```

Lehetséges értékek "FAT", "NTFS" és "CDFS", melyek természetesen a *File Allocation Table* (FAT), a *New Technology File System* (NTFS), illetve a *Compact Disk File System* (CDFS) állományrendszereknek felelnek meg.

A `Path` tulajdonság egy karakterláncot ad vissza, mely a meghajtóbetűből és egy `:` (kettőspont) karakterből áll, például `"A:"`.

A `RootFolder` tulajdonság a meghajtó törzsmappáját adja vissza. A visszaadott érték egy `Folder` objektum.:

```
Dim objFolder
Set objFolder = objDrive.RootFolder
```

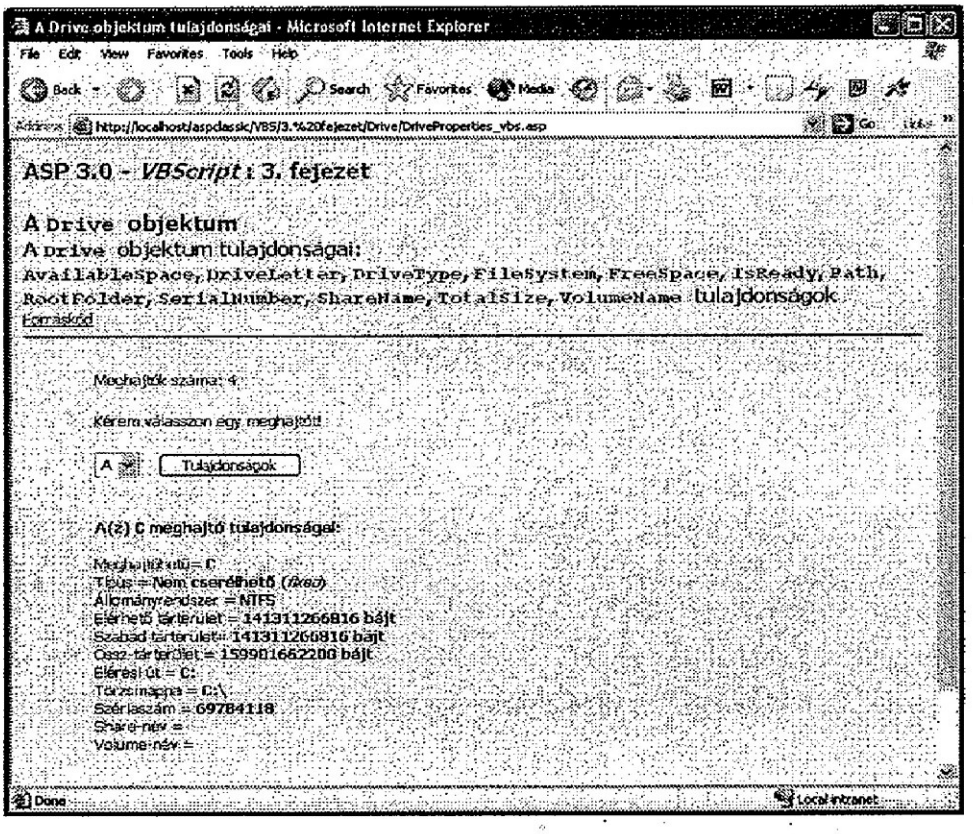
A törzsmappa a meghajtó összes állományát és almappáját tartalmazza.

A `SerialNumber` tulajdonság egy `Long` típusú értékkel a meghajtó szériaszámát adja meg:

```
Dim lngSerialNumber
lngSerialNumber = objDrive.SerialNumber
```

A `ShareName` tulajdonság a távoli vagy hálózati meghajtók esetében a meghajtó megosztott nevét (*share name*) adja meg *Universal Naming Convention* (UNC) formátumban. Lokális meghajtók esetében a visszaadott érték az üres karakterlánc.

```
Dim strShareName
strShareName = objDrive.ShareName
```



3.8. ábra A `Drive` *objektum tulajdonságainak használata*

Az eddig megismert tulajdonságok mind csak olvasható tulajdonságok. A hátramaradt `VolumeName` tulajdonság olvasható és írható is. Értéke egy karakterlánc lehet. Ha leolvassuk a `VolumeName` tulajdonságot, akkor a meghajtó táregységének nevét adja vissza:

```
Dim strVolumeName
strVolumeName = objDrive.VolumeName
```

ha viszont egy értéket rendelünk hozzá, akkor ez lesz a meghajtó táregységének új neve:

```
Dim strVolumeName
strVolumeName = "newname"
objDrive.VolumeName = strVolumeName
```

A `Drive` objektum most megismert tulajdonságainak alkalmazását a `Drive` mappába elhelyezett `DriveProperties_vbs.asp` lap mutatja be. A lapot a 3.8. ábrán láthatjuk.

3.3 A `Folder` objektum

A `FileSystemObject` objektummodell hierarchiában a következő objektum a `Folder` objektum, mely az állományrendszerben a mappának felel meg.

Egy `Folder` objektumot különbözőképpen érhetünk el. Láttuk, hogy a `FileSystemObject` objektum `GetFolder` metódusa egy `Folder` objektumot ad meg és hogy a `Drive` objektum `RootFolder` tulajdonsága is egy `Folder` objektumot ad vissza. A most bemutatásra kerülő `Folder` objektum `SubFolders` tulajdonsága egy `Folder` objektumokból álló gyűjteményt bocsát rendelkezésünkre. Ezen kívül a `Folder` objektum `ParentFolder` tulajdonsága és a következő szakaszban tárgyalásra kerülő `File` objektum szintén `ParentFolder` nevű tulajdonsága is egy `Folder` objektumot ad vissza. Végül emlékezzünk vissza, hogy az úgynevezett speciális mappákat a `FileSystemObject` objektum `GetSpecialFolder` metódusa teszi hozzáférhetővé. Mindezek a tulajdonságok és metódusok megkönnyítik az állományrendszerrel való foglalkozást és lehetővé teszik az állományrendszerben a navigálást bárhonnan bárhova.

Ismerkedjünk meg a `Folder` objektum metódusaival és tulajdonságaival!

3.3.1 A `Folder` objektum metódusai

A `Folder` objektumnak négy metódusa van:

- `Copy`
- `Move`
- `Delete`
- `CreateTextFile`

A `Copy`, `Move` és `Delete` metódusok a `FileSystemObject` objektum `CopyFolder`, `MoveFolder`, illetve `DeleteFolder` metódusaival mutatnak messzemenő hasonla-

tosságot. Mivelhogy a `Copy`, `Move` és `Delete` metódusok egy adott `Folder` objektumhoz, mondjuk `objFolder`, vannak kötve, a `CopyFolder`, `MoveFolder`, illetve `DeleteFolder` metódusokkal ellentétben, nincs szükségük az utóbbiak *source* paraméterére. Ennek a paraméternek a szerepét az `objFolder` objektum tölti be.

A `Copy` metódussal egy mappát másolhatunk egy megadott célhelyre. A metódus szintaxisa a következő:

```
objFolder.Copy(destination [, overwrite])
```

ahol

- *destination* egy karakterlánc, mely a célhelyet adja meg;
- *overwrite* értéke **True** (alapértelmezett érték) vagy **False** lehet. **True** érték esetében az esetleg létező mappa felül lesz írva.

Ha a *destination* utolsó karaktere `"\"`, akkor a *destination* egy létező vagy egy létrehozandó mappát ad meg. A következő szabályok alkalmazhatók:

- ha a *destination* által megadott mappa nem létezik: a `Folder` objektum létrehozza a szükséges mappát és az `objFolder` mappát tartalmával együtt átmásolja az újonnan létrehozott mappába. Ez a leggyakrabban előforduló eset;
- ha a *destination* által megadott célhely egy létező állománynak felel meg: a `Folder` objektum egy hibát generál;
- ha a *destination* által megadott célhely egy létező mappának felel meg: a `Folder` objektum megkísérli az `objFolder` mappát a célmappába másolni. Ha a célmappa `objFolder` mappában tartalmazott mappákkal és/vagy állományokkal azonos nevű mappákat és/vagy állományokat tartalmaz és ha *overwrite* értéke **False**, akkor a `Folder` objektum egy hibát generál;
- ha a *destination* által megadott célhely egy csak olvasható (*read-only*) mappa, akkor a `Folder` objektum egy hibát generál.

A `Move` metódussal egy mappát áthelyezhetünk egy megadott célhelyre. A metódus szintaxisa a következő:

```
objFolder.Move(destination )
```

ahol *destination* egy karakterlánc, mely a célhelyet adja meg.

Ha *destination* utolsó karaktere "\", akkor a *destination* egy létező vagy egy létrehozandó mappát ad meg. A következő szabályok alkalmazhatók:

- ha a *destination* által megadott mappa nem létezik: a Folder objektum létrehozza a szükséges mappát és az objFolder mappát tartalmával együtt áthelyezi az újonnan létrehozott mappába. Ez a leggyakrabban előforduló eset;
- ha a *destination* által megadott célhely egy létező állománynak vagy mappának felel meg: a Folder objektum egy hibát generál.

A Delete metódussal egy mappát törölhetünk. A metódus szintaxisa:

```
objFolder.DeleteFolder( [ force] )
```

ahol *force* értéke **True** vagy **False** (alapértelmezett érték) lehet. **True** érték esetében a csak olvasható (*read-only*) állományok és mappák törölve lesznek.

Ha a Folder objektum nem találja meg a törlendő mappát, akkor egy hibát generál.

A CreateTextFile metódus egy szövegállományt hoz létre és nyit meg. Ugyanakkor az állománynak megfelelő TextStream objektumot adja vissza:

```
Dim objTextStream
Set objTextStream = _
objFolder.CreateTextFile( _
filename [, overwrite [,unicode]])
```

ahol

- *filename* egy karakterlánc, mely a létrehozandó állományt határozza meg;
- *overwrite* értéke **True** vagy **False** (alapértelmezett érték) lehet. **True** érték esetében az esetleg létező állomány felül lesz írva;
- *unicode* értéke **True** vagy **False** (alapértelmezett érték) lehet. **True** érték esetében az állomány Unicode kódban lesz tárolva.

A TextStream objektummal egy külön szakaszban foglalkozunk.

3.3.2 A Folder objektum tulajdonságai

A Folder objektumnak 15 tulajdonsága van:

- Attributes
- DateCreated

- DateLastAccessed
- DateLastModified
- Name
- Path
- ShortName
- ShortPath
- Size
- Type
- IsRootFolder
- Drive
- ParentFolder
- SubFolders
- Files

Ezek közül csak az Attribute és a Name tulajdonságok írhatók és olvashatók, míg a többi csak olvasható.

Az Attribute tulajdonság a mappa attribútumait adja vissza, illetve határozza meg. Értéke egy egész szám lehet:

```
objFolder.Attribute = integer
```

vagy

```
integer = objFolder.Attribute
```

Az Attribute tulajdonság értéke a következő konstansok értékéből adódhat:

érték	konstans	jelentése	az attribútum
0	Normal	normális	
1	ReadOnly	csak olvasható	olvasható/írható
2	Hidden	rejtett	olvasható/írható
4	System	rendszer	olvasható/írható
8	Volume	táregység	csak olvasható
16	Directory	mappa	csak olvasható
32	Archive	archívum	olvasható/írható
1024	Alias	alias	csak olvasható
2048	Compressed	tömörített	csak olvasható

Egy adott Folder objektum esetében az Attribute tulajdonság értéke egyenlő a Folder objektum attribútumainak megfelelő konstansok összegével. Ha például az objFolder objektum egy rejtett rendszermappát reprezentál, akkor attribútumai Hidden (= 2), System (= 4) és Directory (= 16), tehát az

objFolder.Attribute tulajdonság értéke 2 + 4 + 16 = 22. Ha ezt a mappát nem akarjuk elrejteni, ellenben azt akarjuk, hogy ne lehessen a mappában semmit sem változtatni, akkor a Hidden attribútumot el kell távolítanunk, a ReadOnly attribútumot viszont a mappa attribútumaihoz hozzá kell adnunk:

```
objFolder.Attribute = _
objFolder.Attribute - 2 + 1 ' = 21
```

Mivelhogy minden Folder objektum rendelkezik a Directory attribútummal, a Folder objektum Attribute tulajdonságának értéke mindig legalább 16.

Ha egy objFolder mappáról meg akarjuk állapítani, hogy rendelkezik-e egy bizonyos attrib attribútummal, a következőképpen kell eljárnunk: lekérdezzük a mappa Attribute tulajdonságát, majd az eredménnyel és az attrib attribútummal elvégzünk egy **And** műveletet. Ha az eredmény egyenlő az attrib attribútum értékével, akkor az objFolder rendelkezik az attrib attribútummal:

```
If objFolder.Attribute And attrib = attrib Then
  Response.Write _
  "A mappa rendelkezik a kérdéses attribútummal."
End If
```

Könnyebben belátható a fenti eljárás magyarázata, ha egy konkrét esetet vizsgálunk meg. Tételezzük fel, hogy az Attribute tulajdonság által visszaadott érték 6 és tegyük fel, hogy azt szeretnénk ellenőrizni, hogy a megvizsgált mappa rejtett mappa-e vagy nem. A Hidden attribútumnak megfelelő érték 2, tehát minket a 6 **And** 2 érték érdekel. Bináris alakban ez a művelet:

```
110
010
---
010
```

azaz 6 **And** 2 értéke 2, más szóval a vizsgált mappa rejtett. Ha több attribútumot akarunk ellenőrizni, akkor az érdekelt attribútumoknak megfelelő konstansok összegéve hasonlóképpen kell eljárnunk. Ha például az akarjuk ellenőrizni, hogy a mappánk rejtett (Hidden = 2) és csak olvasható (ReadOnly = 1), akkor 6 **And** (2 + 1) = 2 vagy bináris alakban

```
110
011
---
010
```

tehát a mappa nem egyszerre rejtett és csak olvasható (hanem csupán rejtett). Ha végül a Hidden (= 2) és System (= 4) attribútumokból adódó értékkel (2 + 4 = 6 = attrib) végezzük el az összehasonlító műveletet, akkor

```
objFolder.Attribute And attrib = attrib
```

mivelhogy

```
  110
  110
  110
```

ami azt jelenti, hogy a mappa egy rejtett rendszermappa.

A `DateCreated`, `DateLastAccessed` és `DateLastModified` tulajdonságok a mappa létrehozásának, utolsó hozzáférésének, illetve utolsó módosításának dátumát és időpontját adják vissza:

```
Dim dtCreated
Dim dtLastAccessed
Dim dtLastModified

dtCreated = objFolder.DateCreated
dtLastAccessed = objFolder.DateLastAccessed
dtLastModified = objFolder.DateLastModified
```

A `Name`, `Path`, `ShortName` és `ShortPath` tulajdonságok a mappa nevét, abszolút elérési útját, a DOS 8.3 stílusú rövid nevét, illetve a DOS 8.3 stílusú rövid nevű komponensekkel felépített abszolút elérési útját adják meg. A `Name` tulajdonság írható is, a többi csak olvasható:

```
Dim strName
Dim strShortName
Dim strPath
Dim strShortPath

strName = objFolder.Name
strName = "newname"
objFolder.Name = strName
strShortName = objFolder.ShortName
strPath = objFolder.Path
strShortPath = objFolder.ShortPath
```

A `Size` tulajdonság a mappa méretét bájtban kifejezve adja meg:

```
Dim lngBytes
lngBytes = objFolder.Size
```

A mappa mérete egyenlő a mappában létező almappák és állományok méretének összegével.

A `Type` tulajdonság a következő karakterláncok közül egyet ad vissza: `"File Folder"`, `"System Folder"` vagy `"Recycle Bin"`, aszerint, hogy a mappa egy közönséges mappa, egy rendszermappa vagy a lomtár:

```
Dim strType
strType = objFolder.Type
```

Az IsRootFolder tulajdonság értéke **True**, ha a mappa egy meghajtó törzsmappája, különben **False**:

```
Dim blnIsRootFolder
blnIsRootFolder = objFolder.IsRootFolder
```

A Folder objektum mostanáig megismert tulajdonságai különböző típusú értékeket adtak vissza, de egyik értéke sem volt egy objektum. A következő tulajdonságok objektumokat vagy objektumgyűjteményeket, illetve ezek referenciáját adják vissza.

A Drive tulajdonság a mappa meghajtójának megfelelő Drive objektumot adja vissza:

```
Dim objDrive
Set objDrive = objFolder.Drive
```

A ParentFolder tulajdonság a mappa szülőmappájának megfelelő Folder objektumot adja vissza:

```
Dim objParentFolder
Set objParentFolder = objFolder.ParentFolder
```

Ha az objFolder objektum egy meghajtó törzsmappája, akkor a ParentFolder tulajdonság **Nothing** értéket ad vissza.

A SubFolders tulajdonság a mappa almappáinak gyűjteményét adja vissza:

```
Dim colSubFolders
Set colSubFolders = objFolder.Subfolders
```

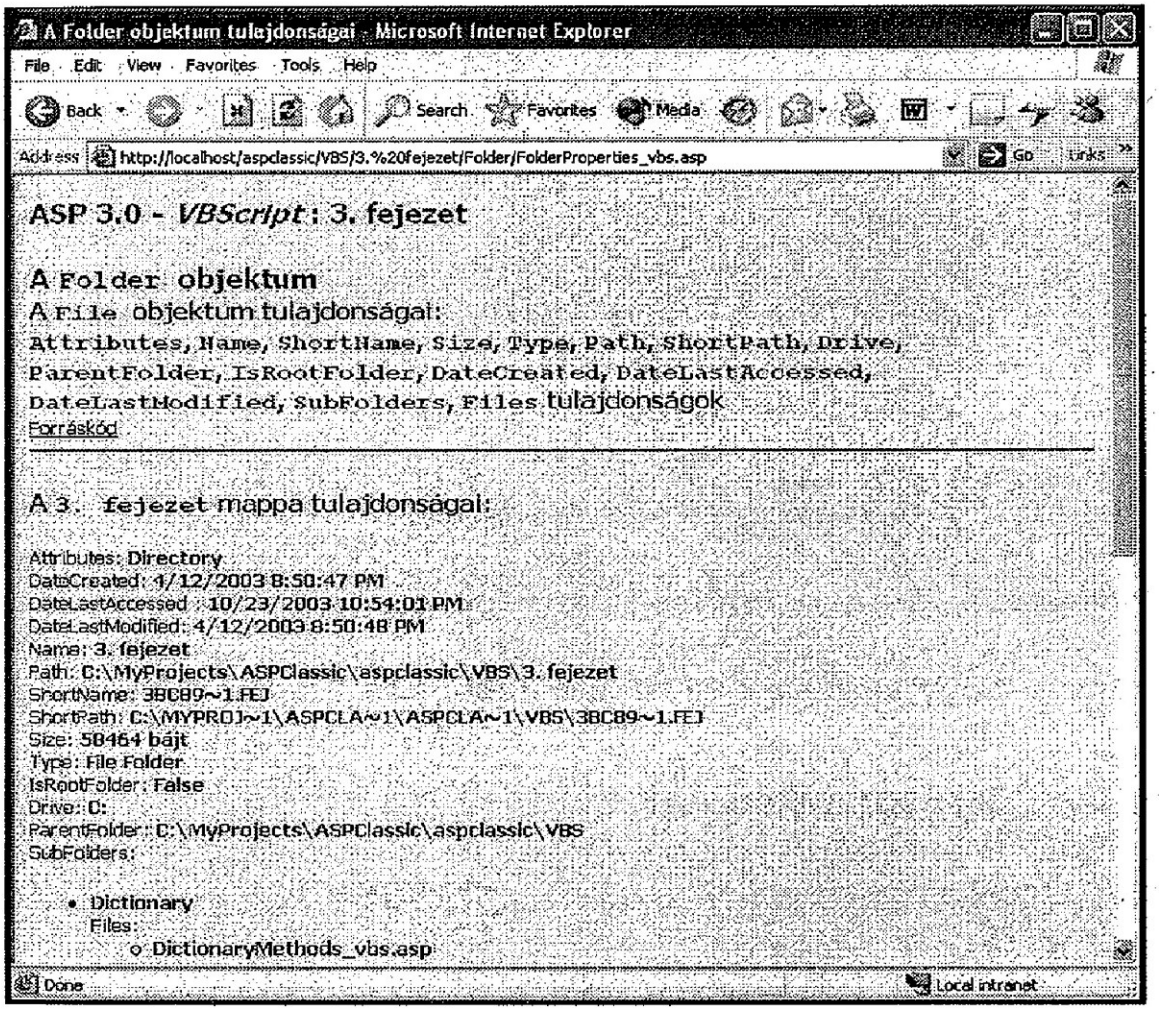
A SubFolders gyűjtemény a gyűjteményeknél megszokott Add metódussal és Item (alapértelmezett tulajdonság) és Count tulajdonságokkal rendelkezik. Az Item metódus paramétereként nem használhatunk indexként egy számot, hanem mindig egy mappaspecifikációt kell megadnunk. A következő kódrészlet megmutatja, miként lehet a SubFolders gyűjtemény Add metódusát és az Item, illetve Count tulajdonságát alkalmazni.

```
<%
 Option Explicit
 Dim objFSO
 Dim objFolder
 Dim colSubFolders

 Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
 Set objFolder = objFSO.GetFolder("C:\Programme")
 Set colSubFolders = objFolder.SubFolders
```

```
Response.Write "Összesen " & _
      colSubFolders.Count & " program van a szerveren.<br>"
If Not objFSO.FolderExists ("C:\Programme\Uj Program") Then
      colSubFolders.Add("Uj Program")
End If
Response.Write "Összesen " & _
      colSubFolders.Count & " program van a szerveren.<br>"
If Not colSubFolders.Item("Internet Explorer") Is Nothing Then
      Response.Write "Az ""Internet Explorer"" " & _
      "mapp a ""C:\Program Files"" mappán belül van."
Else
      Response.Write "Az ""Internet Explorer"" " & _
      "mappa nincs a""C:\Program Files"" mappán belül."
End If
%>
```

A `Files` tulajdonság a mappában található állományok gyűjteményét adja vissza:

```
Dim colFiles
Set colFiles = objFolder.Files
```



3.9. ábra A `Folder` *objektum tulajdonságainak használata*

A `Files` gyűjtemény nem rendelkezik metódussal. Két tulajdonsága van, az `Item` (alapértelmezett tulajdonság) és a `Count` tulajdonság. Itt is, ugyanúgy, mint a `SubFolders` gyűjteménynél az `Item` metódus paramétereként nem használhatunk indexként egy számot, hanem ezúttal egy állományspecifikációt kell megadnunk.

A `Folder` objektum tulajdonságait a `Folder\Folderproperties_vbs.asp` példaalkalmazás mutatja be. A `SubFolders` és `Files` tulajdonságok segítségével az alkalmazás a `VBS\3. fejezet` mappában levő almappákat és az ezekben levő állományokat is felsorolja (3.9. ábra).

3.4 A `File` objektum

A `FileSystemObject` objektummodell egyik nagyon fontos tagja a `File` objektum, mely az állományrendszerben az állománynak felel meg.

Egy `File` objektumot különbözőképpen érhetünk el. Láttuk, hogy a `FileSystemObject` objektum `GetFile` metódusa egy `File` objektumot ad vissza:

```
Dim objFile
Set objFile = objFSO.GetFile(filespec)
```

ahol *`filespec`* egy állományspecifikáció.

Az előző szakaszban megismert `Folder` objektum `Files` tulajdonsága egy `File` objektumokból álló gyűjteményt bocsát rendelkezésünkre. Ez a második lehetőség, hogy hozzáférjünk egy `File` objektumhoz:

```
Dim objFile
Set objFile = objFolder.Files(filename)
```

ahol *`filename`* az `objFolder` mappában levő állomány neve.

A következőkben megvizsgáljuk a `File` objektum metódusait és tulajdonságait.

3.4.1 A `File` objektum metódusai

A `File` objektumnak négy metódusa van:

- `Copy`
- `Move`
- `Delete`
- `OpenAsTextFile`

A `Copy`, `Move` és `Delete` metódusok a `FileSystemObject` objektum `CopyFile`, `MoveFile`, illetve `DeleteFile` metódusaival mutatnak messzemenő hasonlatosságot. Mivelhogy a `Copy`, `Move` és `Delete` metódusok egy adott `File` objektumhoz, mondjuk `objFile`, vannak kötve, a `CopyFile`, `MoveFile`, illetve `DeleteFile` metódusokkal ellentétben, nincs szükségük az utóbbiak *`source`* paraméterére. Ennek a paraméternek a szerepét az `objFile` objektum tölti be.

A `Copy` metódussal egy állományt másolhatunk egy megadott célhelyre. A metódus szintaxisa a következő:

```
objFile.Copy(destination [, overwrite])
```

ahol

- *`destination`* egy karakterlánc, mely a célhelyet adja meg;
- *`overwrite`* értéke **`True`** (alapértelmezett érték) vagy **`False`** lehet. **`True`** érték esetében az esetleg létező állomány felül lesz írva.

Ha a *`destination`* utolsó karaktere `"\"`, akkor a *`destination`* egy létező mappát ad meg. A következő szabályok alkalmazhatók:

- ha a *`destination`* által megadott mappa nem létezik: a `File` objektum egy hibát generál;
- ha a *`destination`* által megadott célhely egy létező állománynak felel meg és *`overwrite`* egyenlő **`True`** értékkel: a `File` objektum egy hibát generál;
- ha a *`destination`* által megadott célhely egy létező mappának felel meg: a `File` objektum megkísérli az `objFile` állományt a célmappába másolni;
- ha a *`destination`* által megadott célhely egy csak olvasható (*read-only*) mappa, akkor az `File` objektum egy hibát generál.

A `Move` metódussal egy állományt áthelyezhetünk egy megadott célhelyre. A metódus szintaxisa a következő:

```
objFile.Move(destination [, overwite])
```

ahol

- *`destination`* egy karakterlánc, mely a célhelyet adja meg;
- *`overwrite`* értéke **`True`** (alapértelmezett érték) vagy **`False`** lehet. **`True`** érték esetében az esetleg létező állomány felül lesz írva.

Ha *destination* utolsó karaktere "\", akkor *destination* egy létező mappát ad meg. A következő szabályok alkalmazhatók:

- ha a *destination* által megadott mappa nem létezik: a File objektum egy hibát generál;
- ha a *destination* által megadott célhely egy létező állománynak vagy mappának felel meg és *overwrite* egyenlő **True** értékkel: a File objektum egy hibát generál.

A Delete metódussal egy állományt törölhetünk. A metódus szintaxisa:

```
objFolder.DeleteFolder( [ force] )
```

ahol *force* értéke **True** vagy **False** (alapértelmezett érték) lehet. **True** érték esetében a csak olvasható (*read-only*) állományok törölve lesznek.

Ha a File objektum nem találja meg a törlendő állományt, akkor egy hibát generál.

A OpenAsTextFile metódus egy szövegállományt nyit meg. Ugyanakkor az állománynak megfelelő TextStream objektumot adja vissza:

```
Dim objTextStream
Set objTextStream = _
objFile.OpenAsTextFile ([, iomode [,format]])
```

ahol

- *iomode* egy egész szám, mely meghatározza a bemenet/kimenet (*input/output*) módját;
- *format* egy egész szám, mely meghatározza az állomány formátumát.

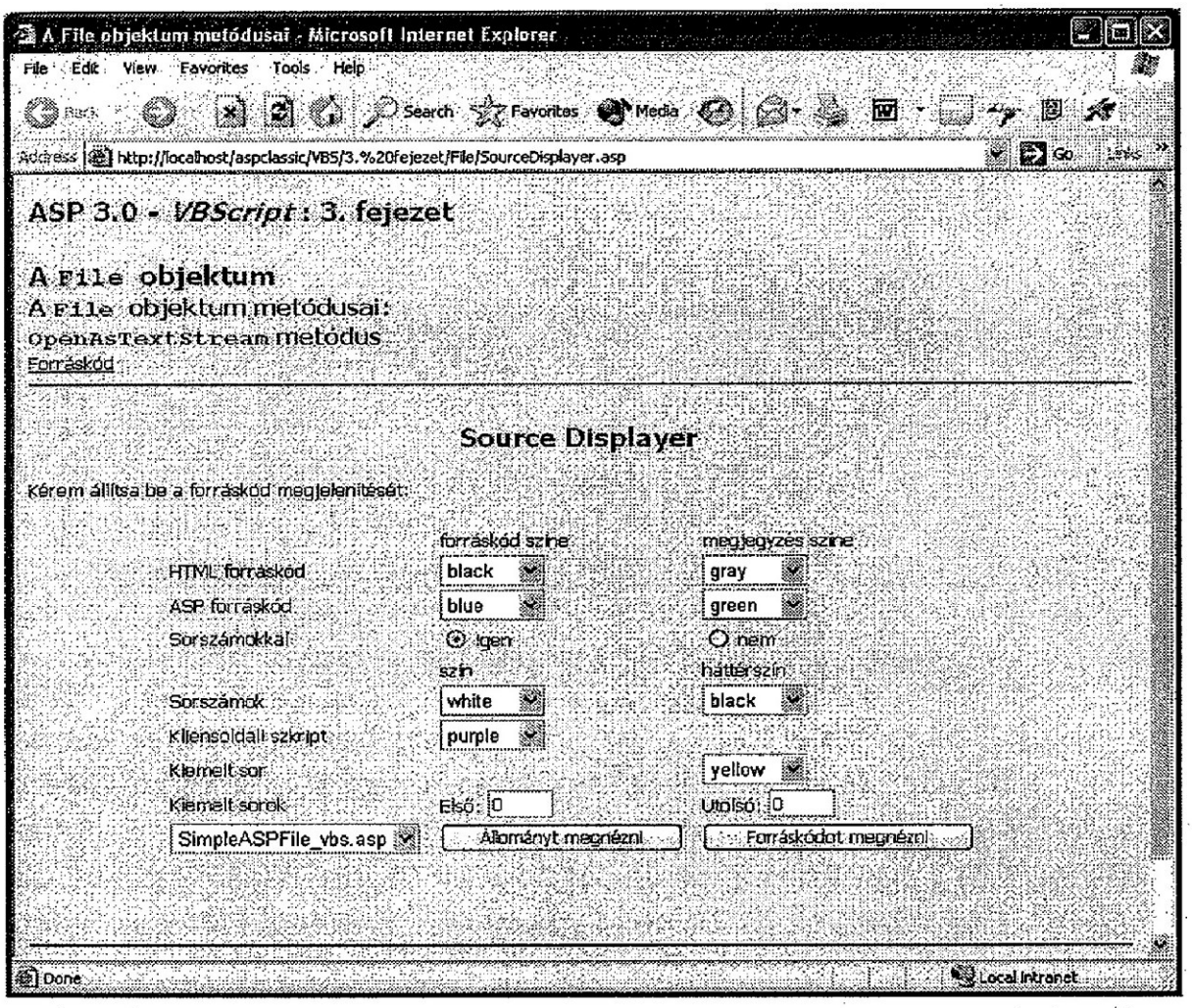
Az *iomode* paraméter értékei a következők lehetnek:

érték	konstans	leírás
1	ForReading	A TextStream objektum segítségével olvashatunk az állományból. Ez az alapértelmezett érték.
2	ForWriting	A TextStream objektum segítségével írhatunk az állományba.
8	ForAppending	A TextStream objektum segítségével az állomány végéhez hozzáfűzhetünk további szöveget.

A `format` paraméter értékei a következők lehetnek:

érték	konstans	leírás
0	`TristateFalse`	ASCII, alapértelmezett formátum
-1	`TristateTrue`	Unicode
-2	`TristateUseDefault`	A rendszer alapértelmezett formátuma

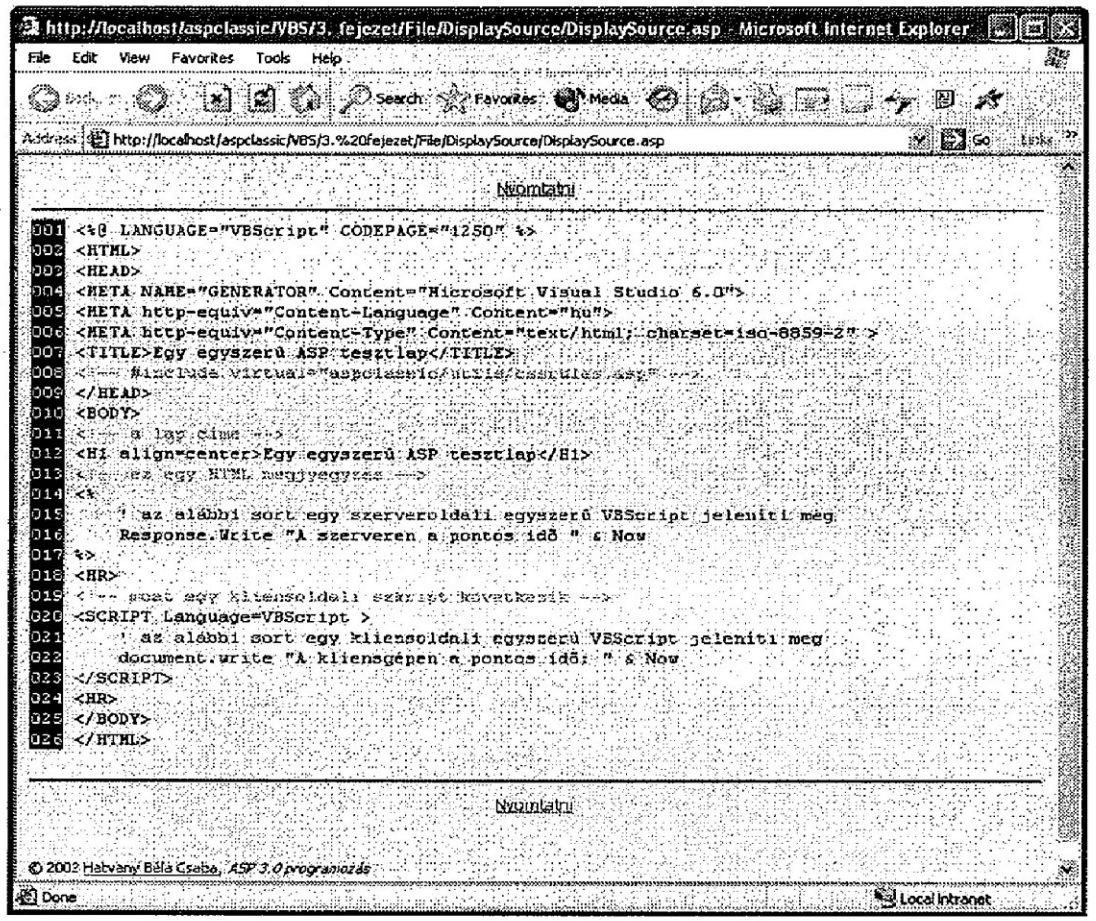
Most nézzük meg a `File` objektum `OpenAsTextFile` metódusának egy érdekes alkalmazását! A `3. fejezet\File\SourceDisplayer.asp` állomány, amit a 3.10. ábrán láthatunk, egy egyszerű alkalmazás kezdőlapja. Az alkalmazás egy ASP vagy HTML lap forráskódját jeleníti meg a böngészőben.



3.10. ábra A `File` *objektum* `OpenAsTextFile` *metódusának alkalmazása*

Az alkalmazás kezdőlapján beállíthatjuk megjelenítés paramétereit: eldönthetjük, hogy a forráskód sorszámokkal vagy sorszámok nélkül legyen megjelenítve, kiválaszthatjuk a HTML és az ASP forráskód-részlet vagy megjegyzés színét, esetleg bizonyos sorokat háttérszínezéssel kiemelhetünk. Az alkalmazás csak két állomány forráskódját jeleníti meg, de könnyen módosítható úgy, hogy tetszőleges HTML vagy ASP állomány

forráskódját megjelenítse. Ugyanis az alkalmazás kezdőlapja egy <FORM> elem segítségével a File\DisplaySource\DisplaySource.asp lapot hívja meg, melynek a megjelenítésre szükséges paramétereket a Request.Form gyűjteménnyel továbbítja. A forráskód megjelenítését ezután a DisplaySource.asp lap végzi el (3.11. ábra).



3.11. ábra A ASP állomány forráskódjának megjelenítése

A forráskód megjelenítésénél a DisplaySource.asp lap figyelembe veszi, hogy az ASP kódrészlet VBScript vagy JScript szkriptnyelven van írva, megkülönbözteti a két szkriptnyelv megjegyzéseit és felismeri a kliensoldali szkripteket is. A mostanáig látott példaalkalmazások forráskódja is a DisplaySource.asp lapban alkalmazott módszerrel lettek megjelenítve. A DisplaySource.asp lapot bőven elláttam megjegyzésekkel, így remélhetőleg könnyen követhető, illetve szükség szerint módosítható. Az állományban használt TextStream objektummal, tulajdonságaival és metódusaival egy külön szakaszban foglalkozunk.

3.4.2 A File objektum tulajdonságai

A File objektumnak 12 tulajdonsága van:

- Attributes

- DateCreated
- DateLastAccessed
- DateLastModified
- Name
- Path
- ShortName
- ShortPath
- Size
- Type
- Drive
- ParentFolder

Ezek közül csak az Attribute és a Name tulajdonságok írhatók és olvashatók, míg a többi csak olvasható.

Az Attribute tulajdonság az állomány attribútumait adja vissza, illetve határozza meg. Értéke egy egész szám lehet:

```
objFile.Attribute = integer
```

vagy

```
integer = objFolder.Attribute
```

Az Attribute tulajdonság értéke ugyanazoknak a konstansoknak az értékéből adódhat, mint a Folder objektum esetében:

érték	konstans	jelentése	az attribútum
0	Normal	normális	
1	ReadOnly	csak olvasható	olvasható/írható
2	Hidden	rejtett	olvasható/írható
4	System	rendszer	olvasható/írható
8	Volume	táregység	csak olvasható
16	Directory	mappa	csak olvasható
32	Archive	archívum	olvasható/írható
1024	Alias	alias	csak olvasható
2048	Compressed	tömörített	csak olvasható

Egy adott File objektum esetében az Attribute tulajdonság értéke egyenlő a File objektum attribútumainak megfelelő konstansok összegével. Ha például az objFile objektum egy rejtett rendszerállományt reprezentál, akkor attribútumai Hidden (= 2), és System (= 4), tehát az objFile.Attribute tulajdonság értéke 2 + 4 = 6. Ha ezt az állományt nem akarjuk elrejteni, ellenben azt akarjuk, hogy ne lehessen az ál-

lományban semmit sem változtatni, akkor a `Hidden` attribútumot el kell távolítanunk, a `ReadOnly` attribútumot viszont az állomány attribútumaihoz hozzá kell adnunk:

```
objFile.Attribute = objFile.Attribute - 2 + 1 ' = 5
```

Ha egy `objFile` állományról meg akarjuk állapítani, hogy rendelkezik-e egy bizonyos `attrib` attribútummal, a következőképpen kell eljárnunk: lekérdezzük az állomány `Attribute` tulajdonságát, majd az eredménnyel és az `attrib` attribútummal elvégzünk egy **And** műveletet. Ha az eredmény egyenlő az `attrib` attribútum értékével, akkor `objFile` rendelkezik az `attrib` attribútummal:

```
If objFile.Attribute And attrib = attrib Then
  Response.Write _
  "Az állomány rendelkezik a kérdéses attribútummal."
End If
```

Könnyebben belátható a fenti eljárás magyarázata, ha egy konkrét esetet vizsgálunk meg. Tételezzük fel, hogy az `Attribute` tulajdonság által visszaadott érték `6` és tegyük fel, hogy azt szeretnénk ellenőrizni, hogy a megvizsgált állomány rejtett állomány-e vagy nem. A `Hidden` attribútumnak megfelelő érték `2`, tehát minket a `6` **And** `2` érték érdekel. Bináris alakban ez a művelet:

```
110
010
---
010
```

azaz `6` **And** `2` értéke `2`, más szóval a vizsgált állomány rejtett. Ha több attribútumot akarunk ellenőrizni, akkor az érdekelt attribútumoknak megfelelő konstansok összegével hasonlóképpen kell eljárnunk. Ha például az akarjuk ellenőrizni, hogy az állomány rejtett (`Hidden = 2`) és csak olvasható (`ReadOnly = 1`), akkor `6` **And** `(2 + 1) = 2`, vagy bináris alakban

```
110
011
---
010
```

tehát az állomány nem egyszerre rejtett és csak olvasható (hanem csupán rejtett). Ha végül a `Hidden` (= `2`) és `System` (= `4`) attribútumokból adódó értékkel (`2 + 4 = 6 = attrib`) végezzük el az összehasonlító műveletet, akkor

```
objFile.Attribute And attrib = attrib
```

mivelhogy

```
110
110
---
110
```

ami azt jelenti, hogy az állomány egy rejtett rendszerállomány.

A `DateCreated`, `DateLastAccessed` és `DateLastModified` tulajdonságok az állomány létrehozásának, utolsó hozzáférésének, illetve utolsó módosításának dátumát és időpontját adják vissza:

```
Dim dtCreated
Dim dtLastAccessed
Dim dtLastModified

dtCreated = objFile.DateCreated
dtLastAccessed = objFile.DateLastAccessed
dtLastModified = objFile.DateLastModified
```

A `Name`, `Path`, `ShortName` és `ShortPath` tulajdonságok az állomány nevét, abszolút elérési útját, a DOS 8.3 stílusú rövid nevét, illetve a DOS 8.3 stílusú rövid nevű komponensekkel felépített abszolút elérési útját adják meg. A `Name` tulajdonság írható is, a többi csak olvasható:

```
Dim strName
Dim strShortName
Dim strPath
Dim strShortPath

strName = objFile.Name
strName = "newname"
objFile.Name = strName
strShortName = objFile.ShortName
strPath = objFile.Path
strShortPath = objFile.ShortPath
```

A `Size` tulajdonság az állomány méretét bájtban kifejezve adja meg:

```
Dim lngBytes
lngBytes = objFile.Size
```

A `Type` tulajdonság egy karakterláncot ad vissza, mely az állomány típusát írja le. Ez a karakterlánc a Windows *registry*-ben van tárolva és a Windows Explorerben az állományoknál a Típus (*Type*) oszlopban jelenik meg:

```
Dim strType
strType = objFolder.Type
```

A `File` objektum mostanáig megismert tulajdonságai különböző típusú értékeket adtak vissza, de egyik értéke sem volt egy objektum. A következő tulajdonságok objektumokat, illetve ezek referenciáját adják vissza.

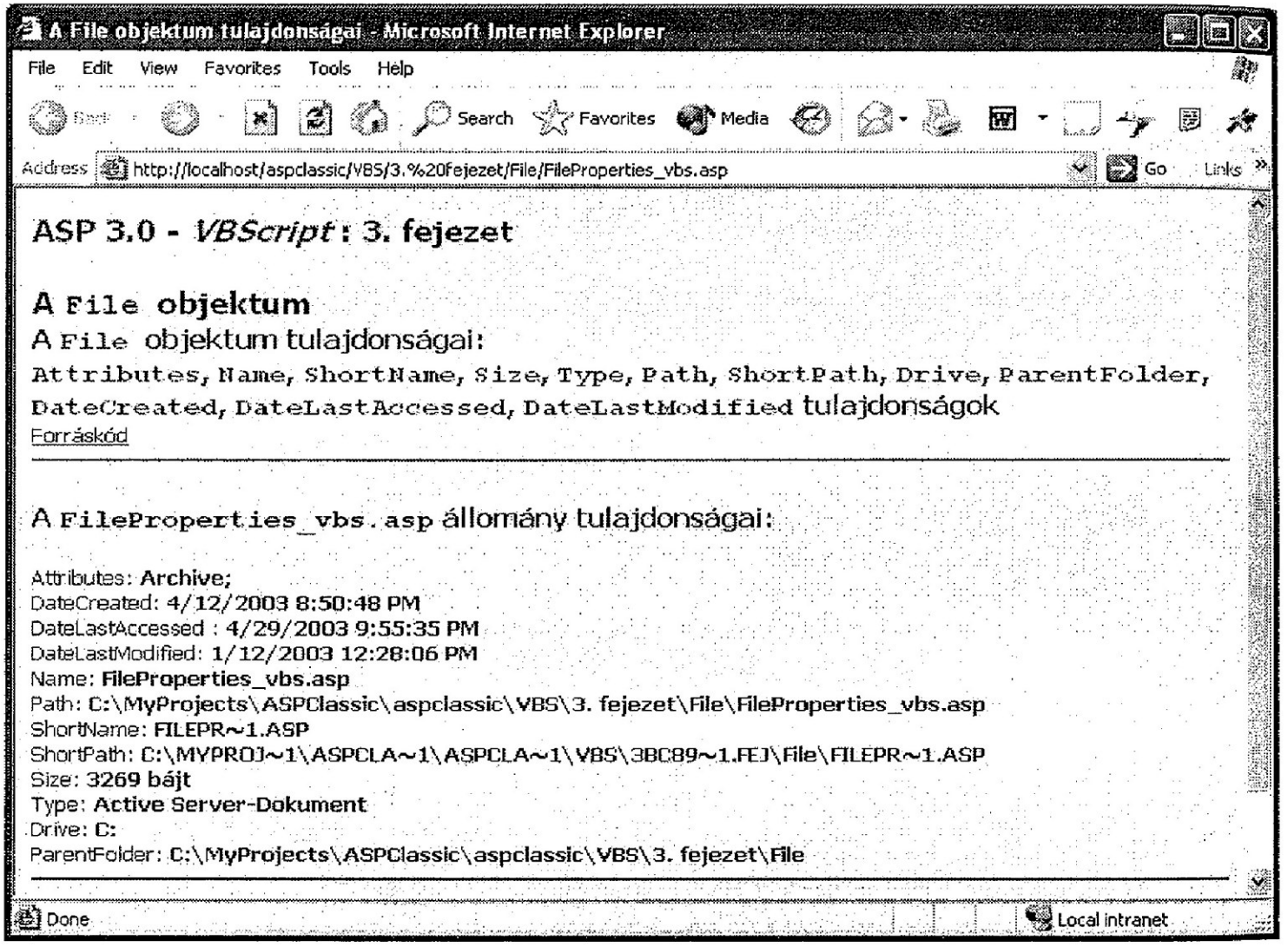
A `Drive` tulajdonság az állomány meghajtójának megfelelő `Drive` objektumot adja vissza:

```
Dim objDrive
Set objDrive = objFile.Drive
```

A `ParentFolder` tulajdonság az állomány szülőmappájának megfelelő `Folder` objektumot adja vissza:

```
Dim objParentFolder
Set objParentFolder = objFile.ParentFolder
```

A `File` objektum tulajdonságait a `File\FileProperties_vbs.asp` példaalkalmazás mutatja be. A 3.12. ábrán ezt a lapot látjuk a böngészőben.



3.12. ábra A `File` objektum tulajdonságainak használata

3.5 A TextStream objektum

A TextStream objektum egy szövegállományt reprezentál. A TextStream objektum segítségével egy szövegállomány tartalmát kiolvashatjuk, illetve egy szövegállományba írhatunk. Mostanáig láttuk, hogy több lehetőségünk van egy létező állományhoz egy TextStream objektumot hozzárendelni vagy egy új állományt létrehozni és egy TextStream objektumot hozzárendelni:

- a FileSystemObject objektum CreateTextFile metódusa egy új szövegállományt hoz létre. Visszatérő értéke egy TextStream objektum, mely a létrehozott szövegállománynak felel meg;
- a FileSystemObject objektum OpenTextFile metódusa egy létező vagy létrehozandó szövegállományt nyit ki. Visszatérő értéke egy TextStream objektum, mely a megnyitott szövegállománynak felel meg;
- a Folder objektum CreateTextFile metódusa egy új szövegállományt hoz létre. Visszatérő értéke egy TextStream objektum, mely a létrehozott szövegállománynak felel meg;
- a File objektum OpenAsTextFile metódusa megnyitja a File objektum által képviselt állományt. Visszatérő értéke egy TextStream objektum, mely a megnyitott szövegállománynak felel meg.

Mindegyik esetben egy TextStream objektum segítségével érhetjük el a szövegállományt. A TextStream objektum metódusai és tulajdonságai lehetővé teszik, hogy az állományból olvassunk vagy hogy az állományba írjunk, illetve további szöveget hozzáfűzzünk.

A TextStream objektum a következő metódusokkal

- Read
- Skip
- ReadLine
- SkipLine
- ReadAll
- Write
- WriteLine
- WriteBlankLines
- Close

és tulajdonságokkal

- AtEndOfLine
- AtEndOfStream

- Column
- Line

rendelkezik. A következő szakaszban ezeket fogjuk közelebbről megvizsgálni.

3.5.1 A TextStream objektum metódusai és tulajdonságai

A Read és Skip metódusok egy-egy *num* **Integer** típusú paramétert várnak, mely meghatározza hány karakterrel végezzen el a metódus egy bizonyos műveletet: a Read metódus karaktereket olvas az állományból, a Skip metódus karaktereket ugrik át. Mindkét metódus használatára az állományt olvasásra kell megnyitnunk, különben egy Bad File Mode hibát idézünk elő.

A Read metódus segítségével megadott számú karaktert olvashatunk egy állományból. A metódus *num* paramétere csak pozitív lehet és meghatározza hány karaktert olvasson a metódus az állományból. Visszatérési értéke a kiolvasott karakterekből álló karakterlánc:

```
Dim strChars
strChars = objTextStream.Read(num)
```

A Skip metódust is csak akkor alkalmazhatjuk, ha az állományt olvasásra nyitottuk meg. A metódus *num* paramétere lehet negatív vagy pozitív egész szám és meghatározza, hány karakterrel tolja el a metódus a kurzor aktuális pozícióját a megnyitott állományon belül. Ha a paraméter negatív, a kurzor az állomány eleje felé lesz eltolva. Ha a paraméter pozitív, a kurzor az állomány vége felé lesz eltolva. A metódus egy üres karakterláncot add vissza:

```
objTextStream.Skip(num)
```

A ReadLine és SkipLine metódusok nem várnak paramétert. A ReadLine metódus egy sort olvas az állományból, a Skip metódus egy sort ugrik át. Mindkét metódus használatára az állományt olvasásra kell megnyitnunk, különben egy Bad File Mode hibát idézünk elő.

A ReadLine metódus egy sort olvas ki az állományból és karakterláncként adja vissza:

```
Dim strLine
strLine = objTextStream.ReadLine
```

A SkipLine metódus egy üres karakterláncot ad vissza:

```
objTextStream.SkipLine
```

A `ReadAll` metódus az egész állományt beolvassa és karakterláncként adja vissza:

```
Dim strAllFile
strAllFile = objTextStream.ReadAll
```

A `ReadAll` metódust is csak olvasásra nyitott állományok esetében alkalmazhatjuk.

A `Write` metódus segítségével egy karakterláncot írhatunk egy állományba. A metódus *str* paramétere megadja az állományba írandó karakterláncot:

```
objTextStream.Write(str)
```

A `WriteLine` metódus **String** típusú *str* paramétere opcionális. A metódus az állományba írja a *str* karakterláncot egy újsor (*newline*) karakterrel megtoldva:

```
objTextStream.WriteLine([str])
```

A `WriteBlankLines` metódus az **Integer** típusú *num* paraméterével megadott *num* darab üres sort ír az állományba:

```
objTextStream.WriteBlankLines(num)
```

A `Write`, `WriteLine` és `WriteBlankLines` metódusok használatára az állományt olvasásra kell megnyitnunk, különben egy `Bad File Mode` hibát idézünk elő. Ha az állományt `ForWriting` paraméterrel nyitottuk meg, akkor ezek a metódusok legelőször az állomány elejére írnak, majd ismételt meghívásukkor az állomány végéhez fűzik a beírandó szöveget. Ha az állományt `ForAppending` paraméterrel nyitottuk meg, akkor ezek a metódusok az állomány végéhet fűzik a beírandó szöveget.

A `Close` metódus bezárja az olvasásra vagy írásra megnyitott állományt:

```
objTextStream.Close
```

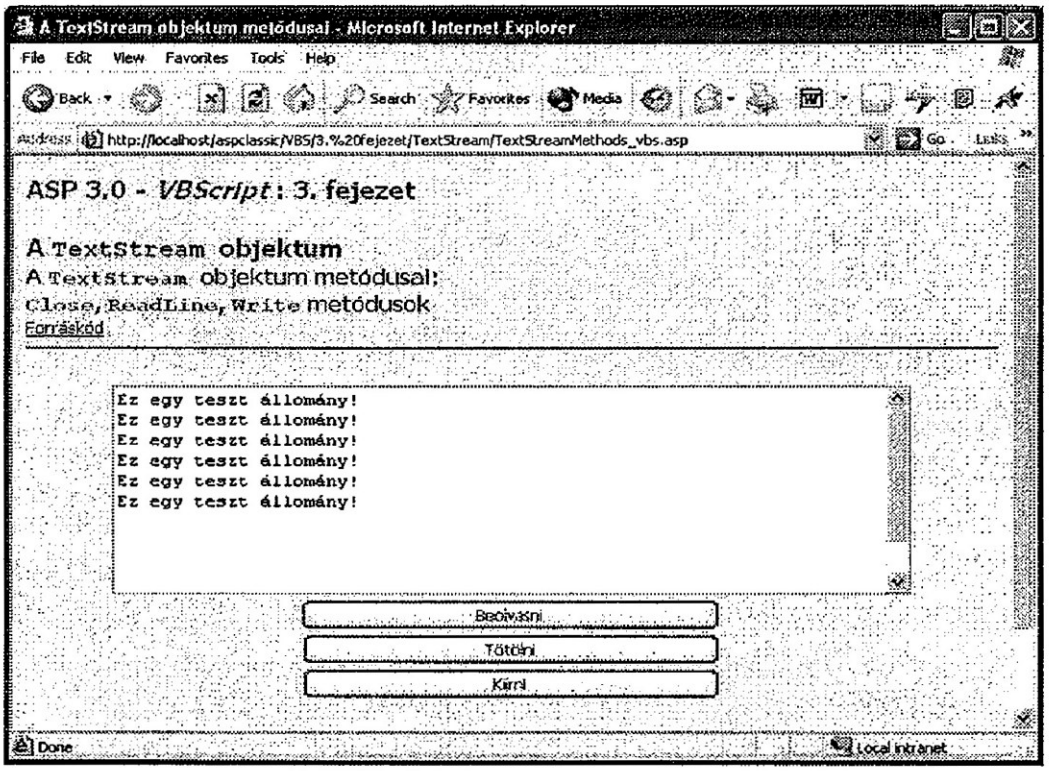
Miután a `Close` metódus bezárta az állományt, a `objTextStream` `TextStream` típusú objektum el lesz távolítva a memóriából. További hivatkozások erre az objektumra hibát idéznek elő.

A `TextStream` objektum tulajdonságai információt szolgáltatnak a megnyitott állományon belül elhelyezett kurzorról. Ezeket a tulajdonságokat lekérdezhetjük, de nem változtathatjuk meg.

Az `AtEndOfLine` és `AtEndOfStream` tulajdonságokat csak olvasásra nyitott állományoknál alkalmazhatjuk. Mindkét tulajdonság egy **Boolean** típusú értéket ad vissza. Az `AtEndOfLine` tulajdonság csak akkor ad **True** értéket vissza, ha a kurzor közvetlen egy újsor karakter előtt van, különben **False** értéket ad vissza. Az

`AtEndOfStream` tulajdonság csak akkor ad **`True`** értéket vissza, ha a kurzor elérte az állomány végét, különben **`False`** értéket ad vissza.

A `Column` és `Line` tulajdonságok a kurzor aktuális pozícióját adják vissza. A `Column` tulajdonság a karakterpozíciót egy sorban adja meg, figyelembe véve, hogy az 1-es karakterpozíció a sor első karakterének felel meg. A `Line` tulajdonság a sorpozíciót az állományon belül adja meg, figyelembe véve, hogy az 1-es sorpozíció az állomány első sorának felel meg.



3.13. ábra A `TextStream` *objektum metódusai*

A `TextStream` most megismert metódusait és tulajdonságait a `TextStream\TestStreamMethods_vbs.asp` példalap alkalmazza. A lapot a 3.13. ábrán láthatjuk. Ha a **Beolvasni** parancsgombra kattintunk, a lap egy `<TEXTAREA>` HTML elembe beolvassa a meglevő `Text.txt` állomány tartalmát. A `<TEXTAREA>` elembe írt szöveget módosíthatjuk, majd a **Kiírni** parancsgombra kattintva kiírhatjuk a módosított szöveget az állományba. Ha most újból beolvassuk az állomány tartalmát a `<TEXTAREA>` elembe, akkor ellenőrizhetjük, mit írt az alkalmazás az állományba.

3.6 A Dictionary objektum

A Dictionary objektumnak különálló szerepe van a Scripting objektumok között, mivelhogy ez az objektum nincs közvetlenül az állományrendszerhez kötve.

A Dictionary objektum adatok tárolására alkalmas. Sok hasonlatosságot mutat a kétdimenziós tömbökkel, azzal a különbséggel, hogy a tárolt adatokat nem egy numerikus indexszel, hanem egy úgynevezett kulcs segítségével érhetjük el. Pontosabban, egy Dictionary objektumban az adatok kulcs/elem (*key/item*) párokként vannak tárolva. Úgy a kulcs, mint az elem is **Variant** típusúak, így egy Dictionary objektumban bármilyen típusú adatot tárolhatunk, bármilyen típusú kulccsal.

Egy Dictionary objektumot a Server objektum CreateObject metódusával:

```
Dim objDictionary
Set objDictionary = _
Server.CreateObject("Scripting.Dictionary")
```

vagy az <OBJECT> elem felhasználásával:

```
<OBJECT RUNAT = "SERVER" SCOPE = "PAGE" PROGID =
"Scripting.Dictionary" ID = "objDictionary">
</OBJECT>
```

hozhatunk létre.

Mindkét esetben egy lapszintű Dictionary objektumot hozunk létre. Ez azt jelenti, hogy a lap letöltése után a Dictionary objektum el lesz távolítva a memóriából. Egyik szerepe a Dictionary objektumnak abban áll, hogy a tárolt adatokat struktúrájával együtt egy másik lapnak adja át. Ilyen esetben a Dictionary objektumot szerver- vagy akár alkalmazásszintű változóba kell tárolnunk. Túl nagy objektumok esetében ez nem tanácsos, mivelhogy az alkalmazás sebessége megsínyli, esetleg teljesen le is fagyhat. (Ezt ellensúlyozandó Microsoft nemrég egy újabb objektumot bocsátott a fejlesztők részére. Bővebb információt a kedves Olvasó ezen a honlapon találhat: http://msdn.microsoft.com/library/en-us/dnvbpj00/html/dictionary.asp.)

A Dictionary objektumnak metódusai:

- Add
- Remove
- RemoveAll
- Exists
- Keys
- Items

tulajdonságai pedig:

- Count
- Key
- Item
- CompareMode

3.6.1 A Dictionary objektum metódusai és tulajdonságai

Az Add metódus hozzáad a Dictionary objektumhoz egy kulcs/elem párt:

```
objDictionary.Add(key, item)
```

ahol

- *key* a kulcsot,
- *item* az elemet adja meg.

A Remove metódus a *key* paraméter által meghatározott elemet távolítja el a Dictionary objektumból:

```
objDictionary.Remove(key)
```

míg a RemoveAll metódus a Dictionary objektum összes elemeit (és következésképpen összes kulcsait is) eltávolítja és így kiüríti a Dictionary objektumot:

```
objDictionary.RemoveAll
```

Az Exists metódussal ellenőrizhetjük, hogy egy megadott kulcsnak megfelel-e egy elem egy Dictionary objektumban. A metódus egy paramétert vár, a keresett elem kulcsát és visszatérő értéke **True**, ha a kulcsnak megfelelően létezik egy elem a Dictionary objektumban, illetve **False** ellenkező esetben:

```
Dim blnResult
blnResult = objDictionary.Exists(key)
```

A Keys és Items metódusok nem rendelkeznek paraméterrel. Visszatérő értékük **Variant** elemekből álló tömbök, melyek a Dictionary objektum kulcsait, illetve elemeit tartalmazzák:

```
Dim arrKeys
Dim arrItems
arrKeys = objDictionary.Keys
arrItems = objDictionary.Items
```

A `Dictionary` objektum `Count` tulajdonsága visszaadja a tárolt kulcs/elem párok számát. A tulajdonság csak olvasható:

```
Dim intCount
intCount = objDictionary.Count
```

Az `Item` tulajdonság visszaadja, illetve meghatározza a *key* kulccsal párosított elemet:

```
Dim varItem
varItem = objDictionary.Item(key)
varItem = item
objDictionary.Item(key) = varItem
```

Ha az `Item` tulajdonsággal a *key* kulcshoz hozzárendeljük az *item* elemet, akkor két lehetőség áll fenn:

- a *key* kulcs az `objDictionary` objektumban párosítva van egy elemmel: ebben az esetben a *key* kulcs az *item* elemmel lesz párosítva;
- a *key* kulcs nem létezik az `objDictionary` objektumban: ebben az esetben a *key*/*item* pár hozzá lesz adva az `objDictionary` objektumhoz.

A `Dictionary` objektum `Key` tulajdonsága felcserél egy adott kulcsot egy másikra:

```
Dim varNewKey
objDictionary.Key(key) = varNewKey
```

Habár egy `Dictionary` objektum kulcsa bármilyen típusú lehet, a gyakorlatban a kulcsokat legtöbbször karakterláncokkal szokták megadni. Ez a módszer egyfelől kevés erőforrást vesz igénybe, másfelől a szöveges kulcsok pontosan leírhatják a megfelelő elemeket. A szöveges kulcsok hátránya a lekérdezésnél lesz észrevehető: pontosan ismerni kell a szöveges kulcs írásmódját. Ezt a nehézséget hivatott kiküszöbölni a `CompareMode` tulajdonság.

A `CompareMode` tulajdonság visszaadja, illetve meghatározza a karakterláncok összehasonlításának módját:

```
Dim intCompareMode
intCompareMode = objDictionary.CompareMode
objDictionary.CompareMode = mode
```

A tulajdonság visszatérő, illetve elfogadott értéke csak a következő lehet:

mode	konstans	leírás
0	`BinaryCompare`	bináris összehasonlítás (különbséget tesz kis- és nagybetű között)
1	`TextCompare`	szöveg-összehasonlítás (nem tesz különbséget kis- és nagybetű között)

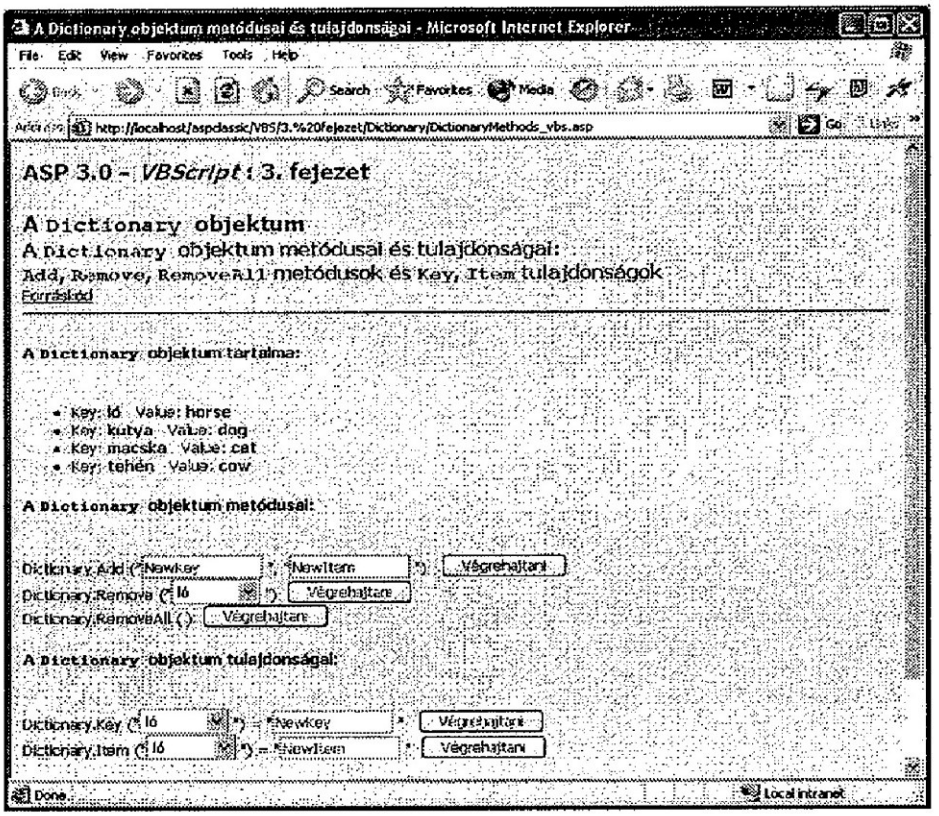
A `Dictionary` objektum alkalmazásaként nézzük meg a `Dictionary` mappában található `DictionaryProperties_vbs.asp` példaalkalmazást!

Az alkalmazást a 3.14. ábrán láthatjuk. A lap felső részében egy négy kulcs/elem párból álló `Dictionary` objektum tartalmát látjuk:

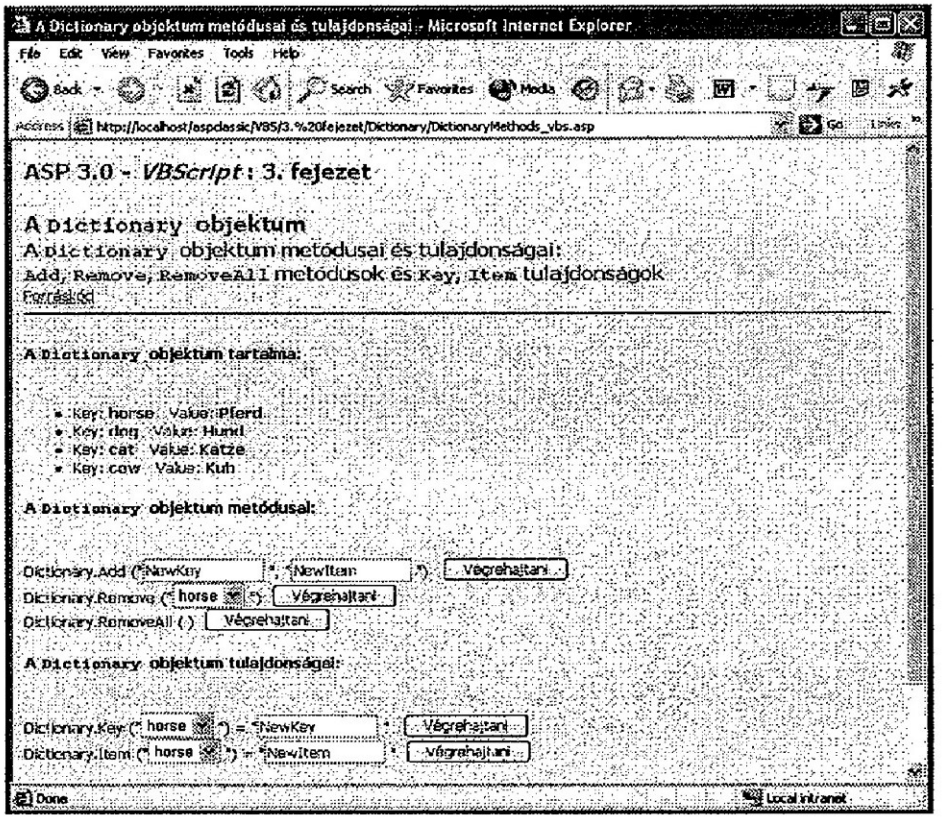
kulcs	elem
"ló"	"horse"
"kutya"	"dog"
"macska"	"cat"
"tehén"	"cow"

A kulcsok és az elemek karakterláncok és a `Dictionary` objektum egy magyar-angol miniszótárt valósít meg.

A lap középső és alsó felében megtaláljuk a `Dictionary` objektum `Add`, `Remove` és `RemoveAll` metódusait, illetve `Key` és `Item` tulajdonságait. A metódusok és tulajdonságok segítségével a magyar-angol szótárt átalakíthatjuk magyar-német vagy akár angol-német szótárrá.



3.14. ábra Magyar-angol szótár mint `Dictionary` *objektum*



3.15. ábra A szótár átalakítása angol-német szótárrá

4. MS web- és email-komponensek

Ha megvizsgáljuk a webalkalmazásokat, akkor hamarosan rájövünk, hogy a legtöbb webalkalmazás sajátos feladatok mellett, alapjában véve ugyanazokat a többé-kevésbé bonyolult feladatokat oldja meg. Az utóbbiak közé sorolhatók a következő feladatok is:

- az alkalmazások egyszerű lapok közötti navigálást kínálnak fel a látogatónak;
- az alkalmazások webhirdetéseket kezelnek;
- az alkalmazások kicserkészik a látogatók böngészőjét és ezekhez igazodva kínálják tartalmukat.
- az alkalmazások email üzenetek elküldését teszik lehetővé.

Természetesen mindezek a feladatok megoldhatók az ASP nyújtotta lehetőségekkel. Mi több, az IIS telepítésével a szerverünkre telepítettük a fenti és más feladatokra szakosított kész megoldásokat. Ezek a megoldások DLL alakban kerültek a gépünkre, más szóval ASP komponensek, vagy pontosabban, telepíthető komponensek (*installable components*). A szóban forgó komponenseket Microsoft webkomponenseinek (*Microsoft Web Components*, MSWC), illetve email-komponenseinek nevezi a szakirodalom. Természetesen más komponensek is elképzelhetők, sőt egy egész komponensgyártó szoftveripar létezik, és amint ezt a következő fejezetben látni fogjuk, mi magunk is készíthetünk ASP komponenseket.

Ebben a fejezetben a Microsoft ASP webkomponenseivel (MSWC) és az email-komponensekkel foglalkozunk. Az MSWC komponensek a következők:

- `AdRotator` (hirdetésforgató) komponens
- `ContentRotator` (tartalomforgató) komponens
- `BrowserType` (a böngésző képességeit megadó, *browser capabilities*) komponens
- `NextLink` (tartalomkapcsoló, *content linking*) komponens
- `Counters` (számlálókat kezelő) komponens
- `IISLog` (naplókezelő, *logging utility*) komponens
- `MyInfo` (egyedi információt kezelő) komponens (Windows XP Professional és az IIS 5.1 nem támogatja ezt a komponenst)
- `PageCounter` (lapszámláló) komponens
- `PermissionChecker` (jogosultság-ellenőrző) komponens
- `Tools` (eszköztár) komponens

az MS email-komponensei pedig a

- CDONTS kollaborációs komponens (Windows XP Professional és az IIS 5.1 nem támogatja ezt a komponenst)
- CDOSYS kollaborációs komponens

A komponensek ASP objektumok sablonjait tartalmazzák. Mindegyik komponens egy objektummodell leírását, az objektum metódusainak, tulajdonságainak és eseményeinek leírását tartalmazza. Pontosabban, a komponenst megtestesítő DLL ezeket az információkat az ASP-nek elérhetővé teszi, azaz feltárja (*expose*). Azért, hogy egy komponenst használhassunk, létre kell hoznunk a komponens által definiált objektum egy példányát. Azt szoktuk mondani, hogy a komponenst instanciáljuk (*instanciate*) vagy példányozzuk.

Egy komponens által definiált objektum példányát hasonlóképpen hozzunk létre, mint a `Scripting` objektumok bármelyikét. Alapjában véve az ASP komponensek semmiben sem térnek el, például a `Scripting` objektumoktól. Mindezek az objektumok az ASP szempontjából külső objektumok és létrehozásuk a `Server` objektum `CreateObject` metódusával történik:

```
Dim objObject
Set objObject = Server.CreateObject(library.class)
```

ahol *library* a könyvtár neve és *class* az objektum neve. Így például a `Scripting` objektumok esetében *library* `Scripting` volt és *class* pedig `FileSystemObject`, `TextStream` vagy `Dictionary` volt. Emlékezzünk vissza, hogy a *library.class* felépítésű paraméter egy karakterlánc. Ezek alapján, az MSWC komponensekkel megadott objektumokat a következőképpen hozhatjuk létre:

```
Dim objMSWC
Set objMSWC = Server.CreateObject("MSWC.compname")
```

Itt `MSWC` a könyvárat adja meg, *compname* pedig a szóban forgó komponens neve.

A fenti állítás, miszerint az MSWC vagy más ASP komponensek nem különböznek alapvetően a `Scripting` komponensektől, még azt is jelenti, hogy a külső komponenseket, ugyanúgy mint a `Scripting` komponenseket, az `<OBJECT>` HTML elem felhasználásával is létrehozhatjuk:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.compname" ID="objMSWC">
</OBJECT>
```

Külső komponenseket mindig jól kell dokumentálnunk. A programozónak ismernie kell az objektummodell architektúráját, a komponens által definiált objektumokat,

ezek metódusait, tulajdonságait és eseményeit, a szükséges paramétereket és ezek típusát és általában mindent, amit a programozásnál az objektumról tudni kell. Egy hiányos dokumentáció egy sikeresen tervezett komponenst teljesen használhatatlanná tehet.

Az MS webkomponenseket részletesen fogjuk tárgyalni. Az MS kollaborációs komponensek részletes bemutatása meghaladja ennek a könyvnek a keretét. Ezekről a komponensekről akár egy egész könyvet is lehet írni. Ugyanakkor fontos a kollaborációs komponensek legalább alapvető használatát ismerni. A fejezet utolsó két szakaszában ezekkel a komponensekkel foglalkozunk.

4.1 Az `AdRotator` (hirdetésforgató) komponens

Az `AdRotator` komponens egy előre megadott ütemrend szerint hirdetéseket forgat: valahányszor a lap frissítve lesz, mindig egy másik hirdetés jelenik meg. A hirdetések legtöbbször digitális képek, úgynevezett szalaghirdetések (*banner*), melyeket a hirdető honlapjához vezető kapcsolattal (*link*) szoktak ellátni.

Az `AdRotator` komponens könyvtára az `adrot.dll` állomány, mely a rendszermappa `system32\inetsrv` almappájában van. Mint láttuk a komponenst a `Server.CreateObject` metódussal:

```
Dim objAdRotator
Set objAdRotator = Server.CreateObject("MSWC.AdRotator")
```

vagy az <OBJECT> HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.AdRotator"
ID=" objAdRotator ">
</OBJECT>
```

hozhatjuk létre.

A komponensnek a helyes működéséhez szüksége van bizonyos információra: hol vannak a képállományok, milyen gyakran jelenjenek meg, hova irányítsa át a böngészőt, miután a látogató egy szalaghirdetésre kattintott stb. Ezeket az információkat a komponens két állományból olvassa ki. Az első állomány az ütemező (*scheduler*) állomány, a második az átirányító állomány. Vegyük sorra ezeket az állományokat!

Az ütemező állomány egy szövegállomány és felépítése a következő:

```
REDIRECT redirectionURL
WIDTH width
HEIGHT height
BORDER border
```

```
*
adURL
adHomePageURL
altText
impression
```

Az állomány két részből áll, melyet egy sor választ el. Az elválasztó sorban csak a * karakter van, pontosan az első helyen. Az állomány első része opcionális, elhagyása esetén az állományt az elválasztó sorral kell megkezdeni. Az állomány első része négy sorból áll, a sorok elején egy-egy kulcsszóval. A kulcsszavak után helyezkednek el a megfelelő értékek. A kulcsszavak jelentése és az értékek típusa a következő:

Kulcsszó	Típus	Leírás
REDIRECT	URL	Az átirányító állomány abszolút vagy relatív URL-je.
WIDTH	szám	A megjelenítendő szalaghirdetés szélessége pixelben kifejezve. Alapértelmezett értéke 440 pixel.
HEIGHT	szám	A megjelenítendő szalaghirdetés magassága pixelben kifejezve. Alapértelmezett értéke 60 pixel.
BORDER	szám	A megjelenítendő szalaghirdetés kerete pixelben kifejezve. Alapértelmezett értéke 1 pixel.

Mint látjuk az első rész az összes hirdetés részére közös tulajdonságokat tartalmaz. Az állomány második része mindegyik hirdetést négy sorral ír le, mindegyik sorban egy-egy tulajdonsággal, szigorúan meghatározott sorrendben. A hirdetések ilyenszerű leírása egymásután következik. A hirdetést leíró négy tulajdonság a következő:

Tulajdonság	Típus	Leírás
adURL	URL	A megjelenítendő szalaghirdetés abszolút vagy relatív URL-je. GIF, JPG vagy PNG képállományok jöhetnek számításba.
adHomePageURL	URL	A hirdetéshez kapcsolódó honlap URL-je. Ha nincs honlap a hirdetéshez kapcsolva, akkor a tulajdonság értéke – (kötőjel).
altText	szöveg	A szalaghirdetés alternatív szövege.
impressions	szám	A megjelenítések gyakoriságát meghatározó egész szám. Ennek a számnak 0 és 4.294.967.295 között kell lennie. Minél nagyobb a szám annál gyakrabban jelenik meg a hirdetés.

Ha az ütemező állományban a hirdetések `i1`, `i2`, ..., `in` gyakorisággal vannak ellátva, akkor az `j`-edik hirdetés megjelenítése az összes megjelenések

$$ij/(i1 + ...+ in) \times 100$$

százalékát teszi ki.

Az ütemező állomány mellett a komponens második segítőállománya az átirányító (*redirector*) állomány. Általában ez az állomány lehet egy DLL könyvtár, egy CGI alkalmazás vagy egy ASP lap. Mi csak az utóbbi esettel foglalkozunk. Esetünkben tehát az átirányító állomány egy ASP lap lesz, mely elvégezi az átirányítást a kívánt honlapra. Amint láttuk, az átirányító állomány URL-jét az ütemező állomány `REDIRECT` tulajdonságával adjuk meg. Az átirányítás a megfelelő *adHomePageURL* tulajdonság által megadott URL-re történik. Ehhez elegendő, ha az átirányító állomány a következő utasítást tartalmazza:

```
Response.Redirect(Request.QueryString("url"))
```

Ezek szerint, az `AdRotator` komponens a `QueryString` gyűjteménnyel juttatja el az átirányító állományhoz a kattintással kiválasztott szalaghirdetésnek megfelelő URL-t. Természetesen az átirányító állományban, még az átirányítás előtt elvégezhetünk más feladatokat is. Legtöbbször ezen a helyen statisztikai számításokat szoktak az alkalmazások elvégezni. Alkalmazásszintű változókba tárolhatjuk a meglátogatott honlapok nézettségét. Az így begyűjtött információt egy naplóba (*logfile*) írhatjuk későbbi feldolgozás céljából vagy esetleg azonnal megjeleníthetjük, mint ahogy a példaalkalmazás során nemsokára látni fogjuk.

Az `AdRotator` komponensnek egy metódusa:

- `GetAdvertisment`

és három tulajdonsága :

- `Border`
- `Clickable`
- `TargetFrame`

van.

A `GetAdvertisment` metódus kiolvassa az ütemező állományból az ott tárolt információt és gondoskodik arról, hogy a megfelelő frekvenciával megjelenítse a szalaghirdetéseket. Úgyszintén a `GetAdvertisment` metódus feladatai közé tartozik a megfelelő URL beillesztése a `QueryString` gyűjteménybe. A metódus a következő szintaxissal rendelkezik:

```
objAdRotator.GetAdvertistment(schedulePath)
```

ahol *schedulePath* az ütemező állomány fizikális elérési útja. A metódus visszaadott értéke a soron levő hirdetést megjelenítő HTML-kódból álló karakterlánc, amit rendszerint a `Response.Write` metódusnak továbbítunk.

A `Border` tulajdonság visszaadja, illetve meghatározza a hirdetést körülvevő keret szélességét. Visszaadott, illetve felvehető értéke egy egész szám, a keret szélessége pixelben kifejezve:

```
Dim intWidth
intWidth = objAdRotator.Border
objAdRotator.Border = intNewWidth
```

Láttuk, hogy az ütemező állomány a `WIDTH` tulajdonsággal határozza meg a hirdetéskeret szélességét. Ezt az értéket a `Border` tulajdonsággal megváltoztathatjuk.

A `Clickable` tulajdonság **`True`** vagy **`False`** értékeket adhat vissza, illetve vehet fel:

```
Dim blnClickable
blnClickable = objAdRotator.Clickable
objAdRotator.Clickable = blnNewClickable
```

A **`True`** érték esetében, ha a látogató a hirdetésre kattint, a komponens az *adHomePageURL* tulajdonsággal meghatározott URL-t juttatja a `QueryString` gyűjteménybe. **`False`** érték esetében a komponens nem továbbit URL-t a `QueryString` gyűjteménynek még akkor sem, ha az *adHomePageURL* tulajdonság meghatároz egyet, azaz nem – (kötőjel).

Végül a `TargetFrame` tulajdonság visszaadja, illetve meghatározza a keret (*frame*) nevét amelyikbe az *adHomePageURL* tulajdonsággal meghatározott URL le lesz töltve. A tulajdonság értéke egy karakterlánc:

```
Dim strFrameName
strFrameName = objAdRotator.TargetFrame
objAdRotator.TargetFrame = strNewFrameName
```

A tulajdonság ugyanazt a szerepet tölti be, mint az `<ANCHOR>` HTML elem `TARGET` attribútuma. Értéke a lapon belül létező bármely keret neve lehet vagy egyike a HTML szokványos keretnevek közül: `"_blank"`, `"_child"`, `"_new"`, `"_parent"`, `"_self"` vagy `"_top"`. Alapértelmezett értéke `"NO FRAME"`.

Nézzünk meg közelebbről egy példaalkalmazást mely az `AdRotator` komponenst alkalmazza!

A `4. fejezet\AdRotator` mappában a következő `AdRotatorSchedule_vbs.txt` szövegállományt találjuk:

```
Redirect AdRotatorRedirector_vbs.asp
Width 300
Height 100
```

```
Border 0
*
imgs/aspfree.gif
http://www.aspfree.com
Itt sok forráskódot és példaalkalmazást találunk.
4
imgs/asptoday.gif
http://www.asptoday.com
Ezernél több ASP cikk várja a látogatót.
2
imgs/aspwire.gif
http://www.aspwire.com
ASP hírek és újdonságok
1
imgs/aspalliance.gif
http://www.aspalliance.com
Itt megismerkedhetünk az ASP.NETtel.
3
imgs/asp101.gif
http://www.asp101.com
Az ASP fejleszt&#245;k honlapja.
1
```

Mint látjuk, az `AdRotator` komponens ütemező állományáról van szó. Az állomány megadja az átirányító állományt: `AdRotatorRedirector_vbs.asp` és négy hirdetést definiál különböző megjelenítési gyakorisággal. A hirdetések képállományai a `4. fejezet\AdRotator\imgs` mappában vannak.

A komponenst az `AdRotatorComponent_vbs.asp` állomány hozza létre. Forráskódjának idevonatkozó része a következő:

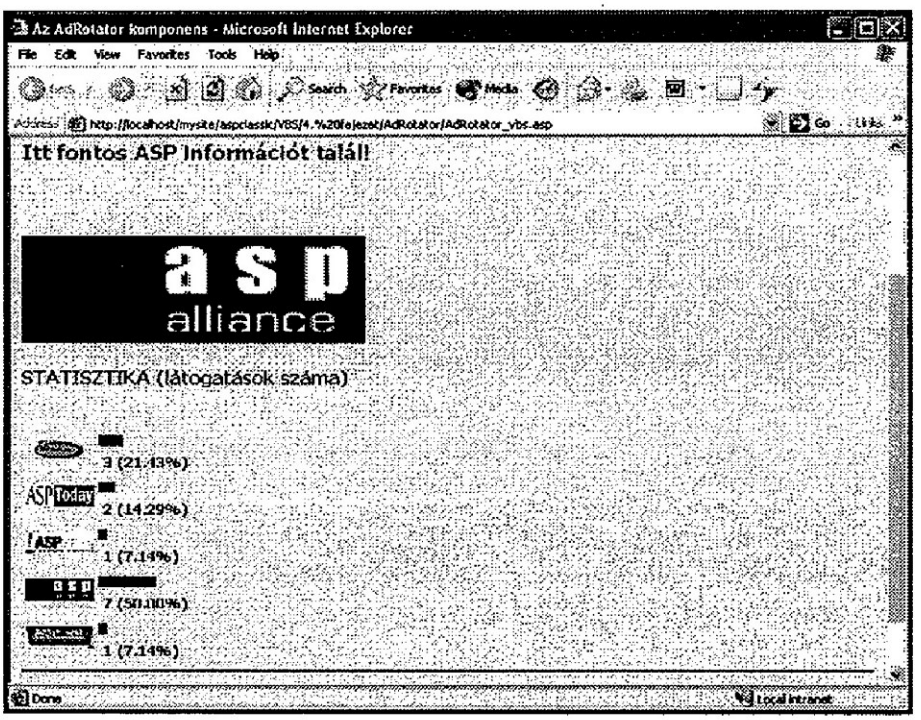
```
<%
 Dim objAdRotator
 Set objAdRotator = Server.Createobject("MSWC.AdRotator")
 ' keret nélkül
 objAdRotator.Border=0
 ' a hirdetés "kattintható"
 objAdRotator.Clickable = True
 ' új ablakba töltjük
 objAdRotator.TargetFrame="_new"
 ' kiírjuk a komponens által generált kódot
 Response.Write(objAdRotator.GetAdvertisement( _
"AdRotatorSchedule_vbs.txt"))
```

A lap alsó részében statisztikai adatokat jelenítünk meg.

Az `AdRotatorRedirector_vbs.asp` átirányító lap forráskódja is igen egyszerű:

```
<HTML>
<BODY>
```

```
<%
 strURL = Request.Querystring("url")

 Application.Lock
 Select Case lcase(strURL)
        Case "http://www.aspfree.com"
             Application("aspfree") = _
             Application("aspfree") + 1
        Case "http://www.asptoday.com"
             Application("asptoday") = _
             Application("asptoday") + 1
        Case "http://www.aspwire.com"
             Application("aspwire") = _
             Application("aspwire") + 1
        Case "http://www.aspalliance.com"
             Application("aspalliance") = _
             Application("aspalliance") + 1
        Case "http://www.asp101.com"
             Application("asp101") = _
             Application("asp101") + 1
 End Select
 Application.Unlock

 Response.Redirect strURL
%>
</BODY>
</HTML>
```



4.1. ábra Az `AdRotator` *komponens működésben*

Minden átirányítás előtt egy alkalmazásszintű változóba mentjük a látogatás számát, amit megfelelően mindig eggyel megemelünk. Mivelhogy a hirdetések honlapját egy

új ablakba töltjük le, az alkalmazás kezdőlapját frissíthetjük. Ekkor az AdRotator komponens forgatja a hirdetést.

A példaalkalmazást a 4.1. ábrán láthatjuk.

4.2 A ContentRotator (tartalomforgató) komponens

A ContentRotator komponens nagyon hasonlít az AdRotator komponensre. A különbség abban áll, hogy a ContentRotator komponens nem képes hirdetéseket, hanem egész HTML tartalmakat forgat. A forgatásra szánt HTML tartalmakat a komponens egy tartalomütemező állományból veszi.

A ContentRotator komponens könyvtára a controt.dll állomány, mely a rendszermappa system32\inetsrv almappájában van. A komponenst a Server.CreateObject metódussal:

```
Dim objContentRotator
Set objContentRotator = _
Server.CreateObject("MSWC.ContentRotator")
```

vagy az <OBJECT> HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.ContentRotator"
ID=" objContentRotator ">
</OBJECT>
```

hozhatjuk létre.

A tartalomütemező (*content schedule*) állomány egy szövegállomány, melynek a felépítése a következő:

```
%% priority // első tartalom
<-- az első tartalom HTML kódja -->
........
%% priority // második tartalom
<-- a második tartalom HTML kódja -->
........
```

ahol *priority* egy egész számú érték, mely a tartalom megjelenítésének gyakoriságát határozza meg. Minél nagyobb az értéke, annál gyakrabban jelenik meg a tartalom. Ha a tartalomütemező állományban a tartalmak p1, p2, ..., pn gyakorisággal vannak ellátva, akkor az i-edik tartalom megjelenítése az összes megjelenések

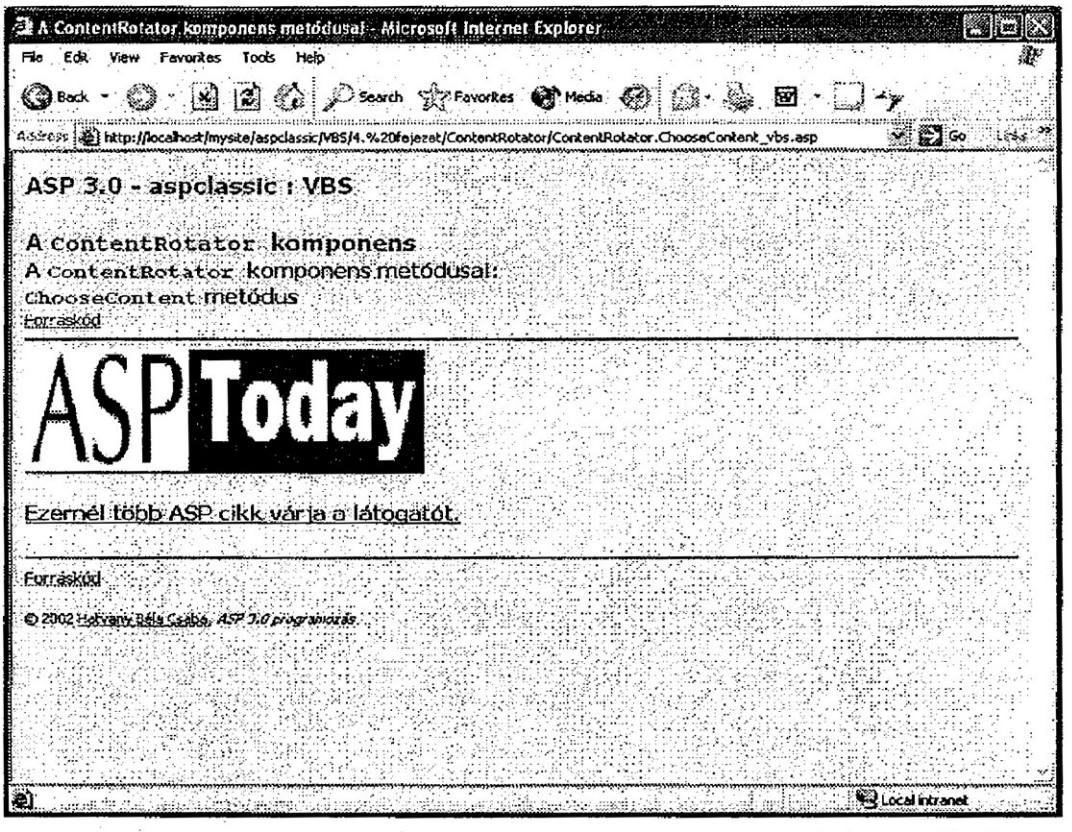
$$pi/(p1 + ...+ pn) \times 100$$

százalékát teszi ki. A megjelenítés gyakorisággát megadó szám után egy C-típusú kommentár helyezhető el. Ezután következik a tartalom szabályos HTML kódja.

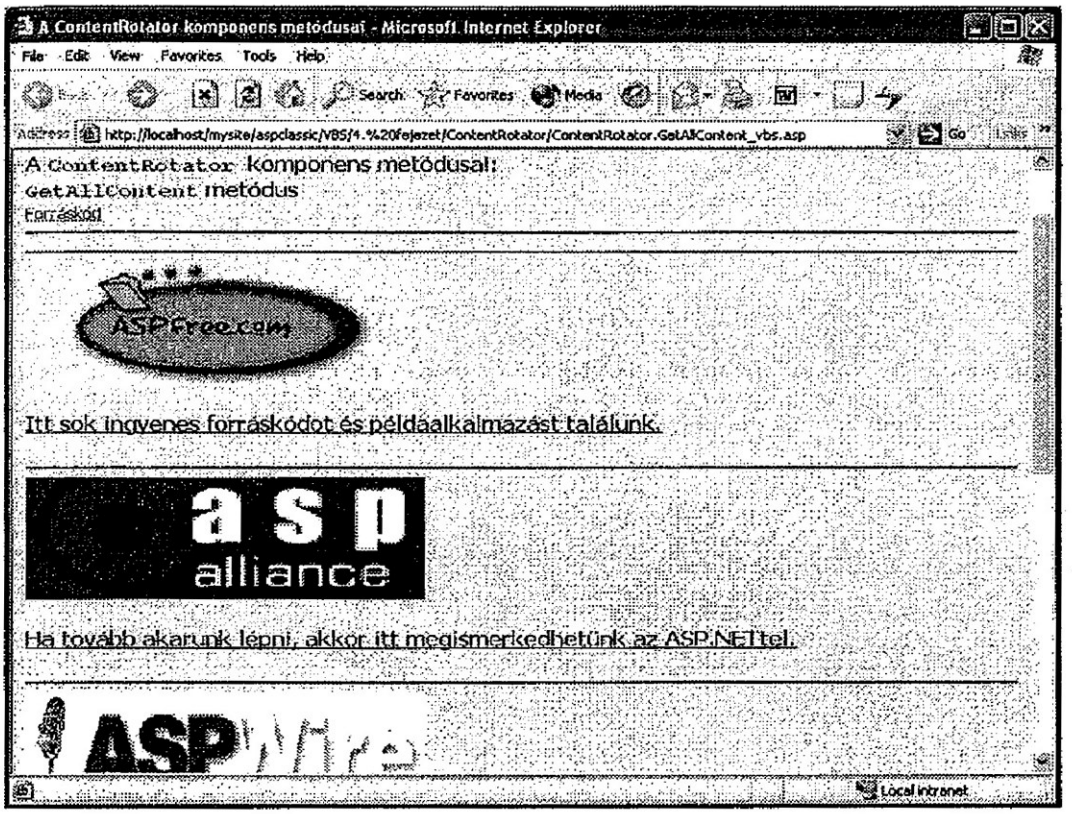
A `ContentRotator` komponensnek két metódusa van: `ChooseContent` és `GetAllContent`. Mindkét metódus egy paramétert vár, a tartalomütemező állomány virtuális vagy fizikai, relatív vagy abszolút elérési útját. Mindkét metódus egy karakterláncot add vissza. A `ChooseContent` metódus esetében ez a karakterlánc a megjelenítésre kerülő tartalom HTML kódja, a `GetAllContent` metódus esetén a karakterlánc az összes tartalom HTML kódja, abban a sorrendben, ahogy a tartalomütemező állományban előfordulnak, két tartalmat egy vízszintes vonallal (`<HR>`) elválasztva.

A `ContentRotator` komponens használatát két példaalkalmazás mutatja be. A `ContentRotator.ChooseContent_vbs.asp` példaalkalmazás forgatja, míg a `ContentRotator.GetAllContent_vbs.asp` példaalkalmazás egyszerre jeleníti meg a tartalomütemezőben megadott tartalmakat. Mindkét alkalmazás ugyanazt a `ContentRotatorSchedule_vbs.txt` tartalomütemező állományt használja.

A két alkalmazást a 4.2. és 4.3. ábrákon láthatjuk.



4.2. ábra A `ContentRotator` *komponens* `ChooseContent` *metódusának használata*



4.3. ábra A `ContentRotator` *komponens* `GetAllContent` *metódusának használata*

4.3 A `BrowserType` (a böngésző képességeit megadó) komponens

A `BrowserType` komponens információt szolgáltat számunkra a kliens böngészőjéről. A komponens segítségével egy ASP alkalmazás figyelembe veheti az aktuális kliens böngészőjét és a böngésző képességeihez igazodhat, azaz böngésző-specifikus lapokat kínálhat fel.

A `BrowserType` komponens a `browscap.dll` DLL könyvtárból áll, mely rendszerint a `winnt\system32\inetsrv` mappába van elhelyezve. Ugyanebben a mappában található a `browscap.ini` szövegállomány, ahonnan a komponens kiolvassa a böngészőre vonatkozó információt.

A `browscap.ini` szövegállomány az összes ismert böngészők leírását tartalmazza. Természetesen ezt az állományt időről időre frissíteni kell. Az állomány legutolsó verzióját a cyScape Inc. http://www.cyscape.com/browscap honlapjáról letölthetjük. (A cyScape Inc. cég a híres **BrowserHawk** böngészőazonosító gyártója és forgalmazója.) Pillanatnyilag a `browscap.ini` állomány 2.5 verziója 2000. február dátummal az aktuális. Ezt az állományt a letöltött ZIP állományban, a `Függelékek` mappában megtalálja a kedves Olvasó.

Tudjuk, hogy az ügyfélgép-kiszolgálógép párbeszédben lekérdezéseivel együtt az ügyfélgép saját magáról is, tehát a böngészőről információt küld a kiszolgálógépnek, többek között a böngésző azonosítóját. A `BrowserType` komponens ezt az azonosítót keresi a szerveren elhelyezett `browscap.ini` szövegállományban, hogy azonosítsa a lekérdező böngészőt.

A `browscap.ini` szövegállomány felépítése a következő:

- `;` (pontvessző, *semicolon*) karakterrel kezdődő sorok megjegyzésnek számítanak;
- az állomány szakaszokra van felosztva: minden egyes szakasz egy böngészőtípust ír le;
- egy szakasz a böngészőazonosítóval kezdődik, amit szögletes zárójelbe kell írni: `[BrowserIndentifier]`;
- egy szakasz addig tart, ameddig egy másik szakasz el nem kezdődik, vagy az állomány végéig;
- egy szakaszon belül a böngésző tulajdonságai (képességei) egy-egy sorban vannak felsorolva *capabilily* = *value* alakban, ahol *capabilily* a képesség elnevezése és *value* a képességet leíró szöveg;
- az állomány sorai legfeljebb 255 karakterből állhatnak.

Egy tipikus szakasz a `browscap.ini` állományból a következőképpen néz ki:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Microsoft Browsers ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;

[IE 1.0]
browser=IE
version=1.0
majorver=#1
minorver=#0
frames=FALSE
tables=FALSE
cookies=FALSE
backgroundsounds=FALSE
vbscript=FALSE
javascript=FALSE
javaapplets=FALSE
platform=Windows95
beta=FALSE
```

Ha a `BrowserType` komponens nem tudja azonosítani a lekérdező böngészőt, akkor a `browscap.ini` állományban megadott alapértelmezett böngésző képességeit használja:

```
;;;;;;;;;;;;;;;;;;;;;;;;
;;; Default Browser    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;
[Default Browser Capability Settings]
browser=Default
Version=0.0
majorver=#0
minorver=#0
frames=False
tables=True
cookies=False
backgroundsounds=False
vbscript=False
javascript=False
javaapplets=False
activexcontrols=False
AK=False
SK=False
AOL=False
beta=False
Win16=False
Crawler=False
CDF=False
AuthenticodeUpdate=
```

A böngészők képességei majdnem minden újabb verzióval bővülnek. A legfontosabbakat a következő táblázatban foglaltam össze:

Képesség megnevezése	Leírás
backgroundsounds	támogat-e háttéri hangállomány
beta	beta verzió
browser	a böngésző neve
cookies	támogat-e sütiket
frames	támogat-e kereteket
javaapplets	támogat-e Java applet-eket
javascript	támogat-e JScript szkripteket
platform	milyen platformon fut
parent	melyik böngészőtől örököl tulajdonságokat
tables	támogat-e táblázatot
vbscript	támogat-e VBScript szkripteket
version	a böngésző verziója

A `BrowserType` komponenst, ugyanúgy mint a többi MSWC komponenst a `Server.CreateObject` metódussal:

```
Dim objBrowserType
Set objBrowserType = Server.CreateObject("MSWC.BrowserType")
```

vagy az <OBJECT> HTML elem segítségével:

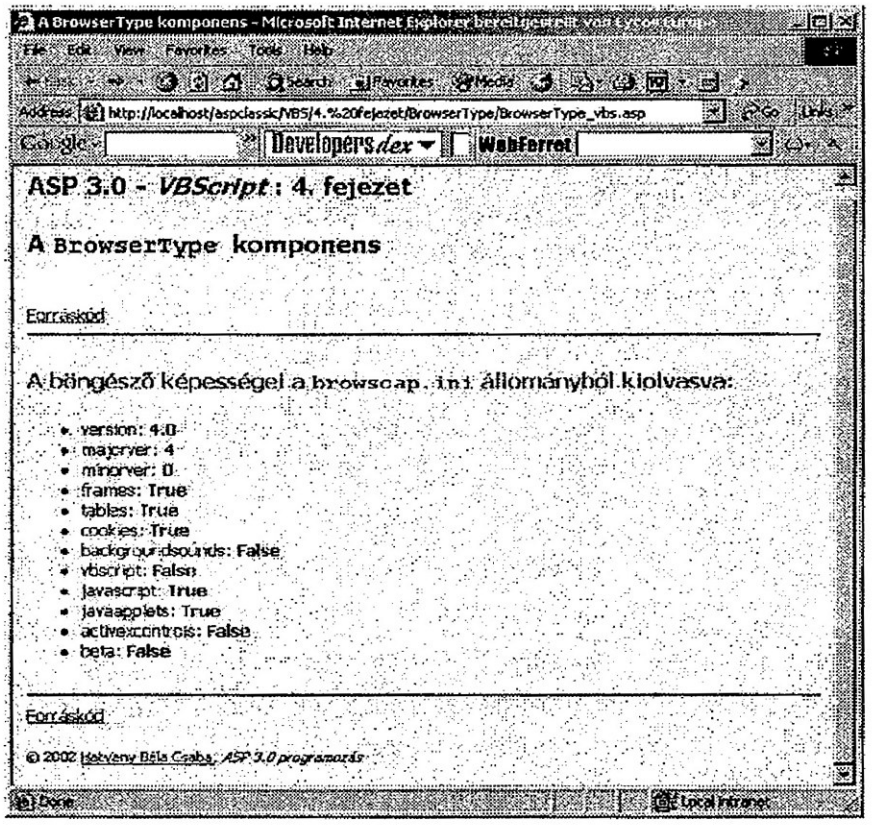
```
<OBJECT RUNAT="SERVER" PROGID="MSWC.BrowserType "
ID=" objBrowserType ">
</OBJECT>
```

hozhatjuk létre.

A komponensnek csupán egy metódusa van. A `Value` metódus visszaadja a paraméterével megnevezett tulajdonság értékét:

```
Dim val
val = objBrowserType(capability)
```

A `4. fejezet/BrowserTyp/BrowserType_vbs.asp` példaalkalmazás illusztrálja a `BrowserType` komponens használatát (4.4. ábra).



4.4. ábra A `BrowserType` *komponens kiolvassa a* `browscap.ini` *állományból a böngésző képességeit*

A browscap.ini állományban megadott tulajdonságok a böngésző típusára vonatkoznak és nem szolgáltatnak semmilyen információt az ügyfélgépen használt beállításáról. Sok esetben azonban ilyennemű információ igen hasznos lehet az ASP alkalmazás részére. Például némelykor nem közömbös, hogy milyen színmélységre van a kliens gépe beállítva. Az Internet Explorer 5.0 vagy későbbi böngészők hozzáférhetővé teszik ezeket az információkat is a szerver részére. Az új lehetőséggel viselkedést

(*behavior*) leíró komponenseket csatolhatunk egy HTML elemhez. Ezeket a különleges komponenseket, melyeket különben Visual Basic, C++, XML vagy szkriptekkel lehet előállítani viselkedésnek (*behavior*) nevezzük.

Ennek a könyvnek nem célja a viselkedések részletes tárgyalása. Az témában érdekelt Olvasónak az MSDN Library idevágó részeit javaslom további információ forrásként. Az MSDN Library idevágó *online* honlapját, illetve az *offline* MSDN Library 2002. április kiadás idevágó részét a letöltött ZIP állományban levő HTML lapról könnyen elérhetjük. Szintén innen érhető el Dino Esposito kitűnő előadásának PowerPoint prezentációja.

Mi az Internet Explorer 5.0 vagy későbbi által rendelkezésünkre bocsátott egyik viselkedés, a `clientCaps` viselkedést fogjuk használni. Ez a viselkedés összesen tizenhárom böngésző tulajdonságot képes megállapítani és egy sütibe írni. A sütibe tárolt információt ezután a `BrowserType` komponens el tud érni és az ASP alkalmazás rendelkezésére bocsátani.

A `clientCaps` viselkedés által kiolvasott tulajdonságok a következők:

Tulajdonság	Leírás
`availHeight`	a képernyő magassága pixelben, dekorációk nélkül
`availWidth`	a képernyő szélessége pixelben, dekorációk nélkül
`height`	a képernyő magassága pixelben, dekorációkkal együtt
`width`	a képernyő szélessége pixelben, dekorációkkal együtt
`bufferDepth`	a kliensgép puffermélysége
`colorDepth`	a kliensgép színmélysége
`connectionType`	a hálózati kapcsolat típusa
`cookieEnabled`	értéke **`True`**, ha a kliensgép fogadja a sütiket, különben **`False`**
`cpuClass`	a kliensgép processzortípusa
`javaEnabled`	értéke **`True`**, ha a kliensgép támogatja a Java kódot, különben **`False`**
`platform`	az operációs rendszer
`systemLanguage`	az operációs rendszer nyelve
`userLanguage`	a böngésző nyelve

A `clientCaps` viselkedést a következőképpen használhatjuk. Létrehozunk egy HTML lapot, melyben egy sajátos (XML) elemhez hozzákapcsoljuk a `clientCaps` viselkedést a következő `STYLUS` attribútummal:

```
<IECAPS ID="objClientCaps"
STYLE="behavior:url(#default#clientCaps)"/>
```

A fenti példában az XML elem neve `IECAPS`, ezt tetszőlegesen választhatjuk. Az `ID` attribútum a viselkedés (pontosabban a viselkedésnek megfelelő komponens létrehozandó példányának) nevét határozza meg. Ezt is tetszésünk szerint választhatjuk. A `STYLE` attribútum értéke viszont mindig `"behavior:url(#default#clientCaps)"` legyen.

A HTML állománynak ezen kívül egy kliensoldali <SCRIPT> elemben egy függvényt kell definiálnia, mely a viselkedés által kiolvasott tulajdonságokat egy sütibe írja. Ezt a függvényt a lap letöltésekor kell végrehajtani.

Mi a következő `ClientCaps.htm` HTML állományt fogjuk használni:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function createCookie() {
    var strCookie = new String();
    strCookie = 'width=' + objClientCaps.width
              + '&height=' + objClientCaps.height
              + '&availWidth=' +
              objClientCaps.availWidth
              + '&availHeight=' +
              objClientCaps.availHeight
              + '&bufferDepth=' +
              objClientCaps.bufferDepth
              + '&colorDepth=' +
              objClientCaps.colorDepth
              + '&javaEnabled=' +
              objClientCaps.javaEnabled
              + '&cookieEnabled=' +
              objClientCaps.cookieEnabled
              + '&connectionType=' +
              objClientCaps.connectionType
              + '&platform=' + objClientCaps.platform
              + '&cpuClass=' + objClientCaps.cpuClass
              + '&systemLanguage=' +
              objClientCaps.systemLanguage
              + '&userLanguage=' +
              objClientCaps.userLanguage;

    document.cookie = 'BrowsCaps=' + strCookie;

  }
</SCRIPT>
</HEAD>
<BODY ONLOAD="createCookie();">
<IECAPS ID="objClientCaps"
STYLE="behavior:url(#default#clientCaps)"/>
</BODY>
</HTML>
```

Amint látjuk, a `createCookie` függvény egy `BrowsCaps` nevű sütit hoz létre, mely karakterláncot tartalmaz. A karakterlánc `&` karakterrel elválasztott *tulajdonság=érték* párokkal írja le a tulajdonságokat. A függvényt a lap letöltéskor hívjuk meg:

```
<BODY ONLOAD="createCookie();">
```

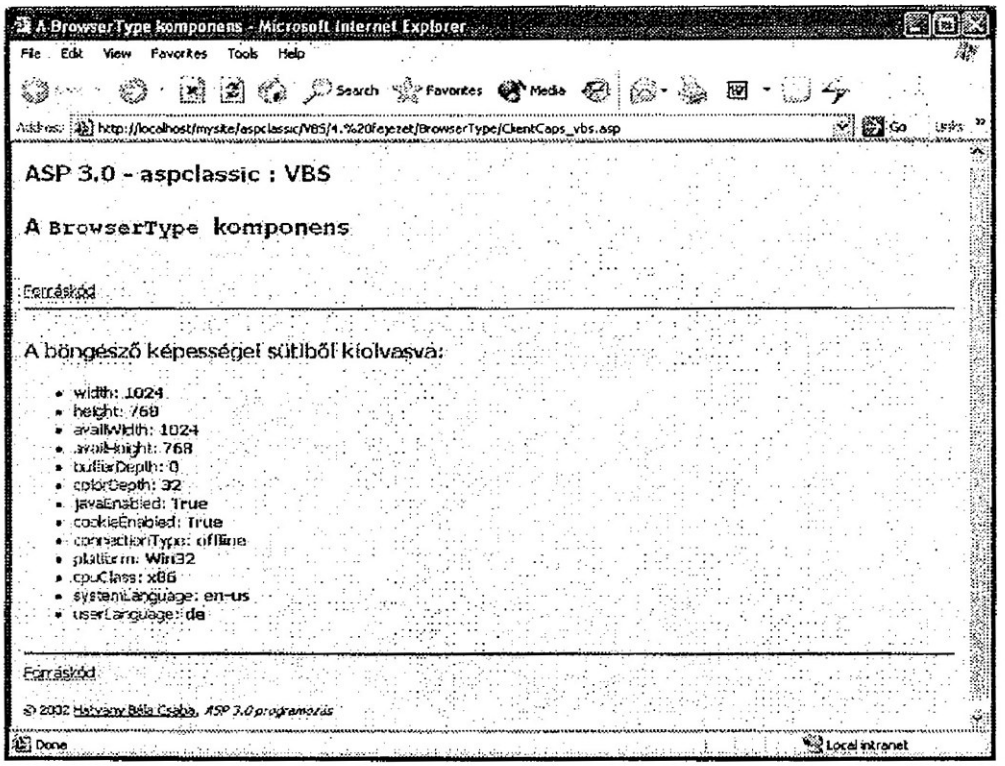
Azért, hogy a sütit egy ASP lapban felhasználhassuk, egy METADATA rendelkezéssel a laphoz kötjük:

```
<!--METADATA TYPE="Cookie" NAME="BrowserCaps"
SRC="ClientCapsBehavior/ClientCaps.htm"-->
```

Itt a `NAME` megadja a süti nevét és az `SRC` a sütit létrehozó HTML lap elérési útját. Ezután létrehozhatunk egy `BrowserType` komponenst és kiolvashatjuk a sütiből a tulajdonságokat. Példaalkalmazásunkban ezt a feladatot a 4. fejezetnek megfelelő mappában elhelyezett `BrowserType/ClientCapsBehavior_vbs.asp` állomány végzi el:

```
<H2>A böngésző képességei sütiből kiolvasva:</H2>
<%
 Dim objBrowserType
 ' létrehozunk egy BrowserType komponenst
 Set objBrowserType = _
Server.CreateObject("MSWC.BrowserType")
%>
<!--METADATA TYPE="Cookie" NAME="BrowserCaps"
SRC="ClientCapsBehavior/ClientCaps.htm"-->
<UL>
<LI>width: <B><% = objBrowserType.width %></B></LI>
<LI>height: <B><% = objBrowserType.height %></B></LI>
<LI>availWidth: <B><% = objBrowserType.availWidth %></B></LI>
<LI>availHeight: <B><% = objBrowserType.availHeight %></B></LI>
<LI>bufferDepth: <B><% = objBrowserType.bufferDepth %></B></LI>
<LI>colorDepth: <B><% = objBrowserType.colorDepth %></B></LI>
<LI>javaEnabled: <B><% = objBrowserType.javaEnabled %></B></LI>
<LI>cookieEnabled: <B><% = objBrowserType.cookieEnabled
%></B></LI>
<LI>connectionType: <B><% = objBrowserType.connectionType
%></B></LI>
<LI>platform: <B><% = objBrowserType.platform %></B></LI>
<LI>cpuClass: <B><% = objBrowserType.cpuClass %></B></LI>
<LI>systemLanguage: <B><% = objBrowserType.systemLanguage
%></B></LI>
<LI>userLanguage: <B><% = objBrowserType.userLanguage %></B></LI>
</UL>
```

Az eredményt a 4.5. ábrán láthatjuk.



4.5. ábra A `BrowserType` *komponens sütiből kiolvassa a kliensgép képességeit*

4.4 A `NextLink` (tartalomkapcsoló) komponens

A `NextLink` komponens segítségével több HTML vagy ASP lapot vagy más, böngészőktől támogatott állományt, mint például kép- vagy XML-állományt, egy tartalomjegyzékbe foglalhatunk össze, illetve köztük könnyen navigálhatunk.

A komponens által kezelt lapokról a szükséges információt egy szövegállományba kell elhelyeznünk, az úgynevezett tartalomjegyzékbe (*content list*). A szövegállomány minden egyes sora egy lapnak felel meg:

```
pageURL description  comment
```

ahol

- `pageURL` a lap virtuális vagy fizikális relatív URL-je. Abszolút URL címek nem engedélyezettek;
- `description` a tartalomjegyzékben megjelenítendő szöveg vagy lapleírás, melynek kapcsolata (*link*) a megfelelő `pageULR` által megadott laphoz vezet;
- `comment` egy opcionális megjegyzés;
- a `pageURL`, `description` és `comment` elemeket tabulátorokkal kell elválasztani, nem közönséges szóközökkel.

A később bemutatásra kerülő példában a következő `toc.txt` tartalomjegyzéket használtam:

```
Pages/page1.asp      Els&#245; oldal
Pages/page2.asp      Második oldal
Pages/page3.asp      Harmadik oldal
Pages/page4.asp      Negyedik oldal
Pages/page5.asp      Ötödik oldal
Pages/page6.asp      Hatodik oldal
Pages/page7.asp      Hetedik oldal
Pages/page8.asp      Nyolcadik oldal
Pages/page9.asp      Kilencedik oldal
Pages/page10.asp     Tizedik oldal
```

A `NextLink` komponens DLL könyvtára a `nextlink.dll` állomány, mely rendszerint a `winnt\system32\inetsrv` mappában található. A `NextLink` komponenst, ugyanúgy, mint a többi MSWC komponenst a `Server.CreateObject` metódussal:

```
Dim objNextLink
Set objNextLink = Server.CreateObject("MSWC.NextLink ")
```

vagy az <OBJECT> HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.NextLink "
ID=" objNextLink ">
</OBJECT>
```

hozhatjuk létre.

A komponens a következő metódusokkal:

- `GetListCount`
- `GetListIndex`
- `GetPreviuosURL`
- `GetPreviousDescription`
- `GetNthURL`
- `GetNthDescription`
- `GetNextURL`
- `GetNextDescription`

és egy tulajdonsággal

- `About`

rendelkezik.

A `GetListCount` metódus paramétere a tartalomjegyzék elérési útját várja és a tartalomjegyzékben felsorolt lapok számát adja vissza:

```
Dim intListCount
intListCount = objNextLink.GetListCount(tocPath)
```

A `GetListIndex` a tartalomjegyzékből megjelenített lap sorszámát adja vissza. A metódus egy paramétert vár, a tartalomjegyzék elérési útját:

```
Dim intListIndex
intListIndex = objNextLink.GetListIndex(tocPath)
```

Szintén egy-egy paramétert várnak a `GetPreviousURL`, `GetPreviousDescription`, `GetNextURL`, `GetNextDescription` metódusok, a tartalomjegyzék elérési útját. Mindegyikük egy karakterláncot ad vissza, nevezetesen az aktuális lap a tartalomjegyzék szerint előtti, illetve következő lap URL-jét vagy leírását:

```
Dim strURL
Dim strDesc
strURL = objNextLink.GetPreviousURL(tocPath)
strDesc = objNextLink.GetPreviousDescription(tocPath)
strURL = objNextLink.GetNextURL(tocPath)
strDesc = objNextLink.GetNextDescription(tocPath)
```

Ha az aktuális lap a tartalomjegyzék első lapja, akkor `GetPreviousURL` és `GetPreviousDescription` a tartalomjegyzék utolsó lap adatait adják vissza. Ha az aktuális lap nem szerepel a tartalomjegyzékben, akkor ezek a metódusok a tartalomjegyzék első lapjának adatait adják vissza.

Ha az aktuális lap a tartalomjegyzék utolsó lapja, akkor `GetNextURL` és `GetNextDescription` a tartalomjegyzék első lap adatait adják vissza. Ha az aktuális lap nem szerepel a tartalomjegyzékben, akkor ezek a metódusok a tartalomjegyzék utolsó lapjának adatait adják vissza.

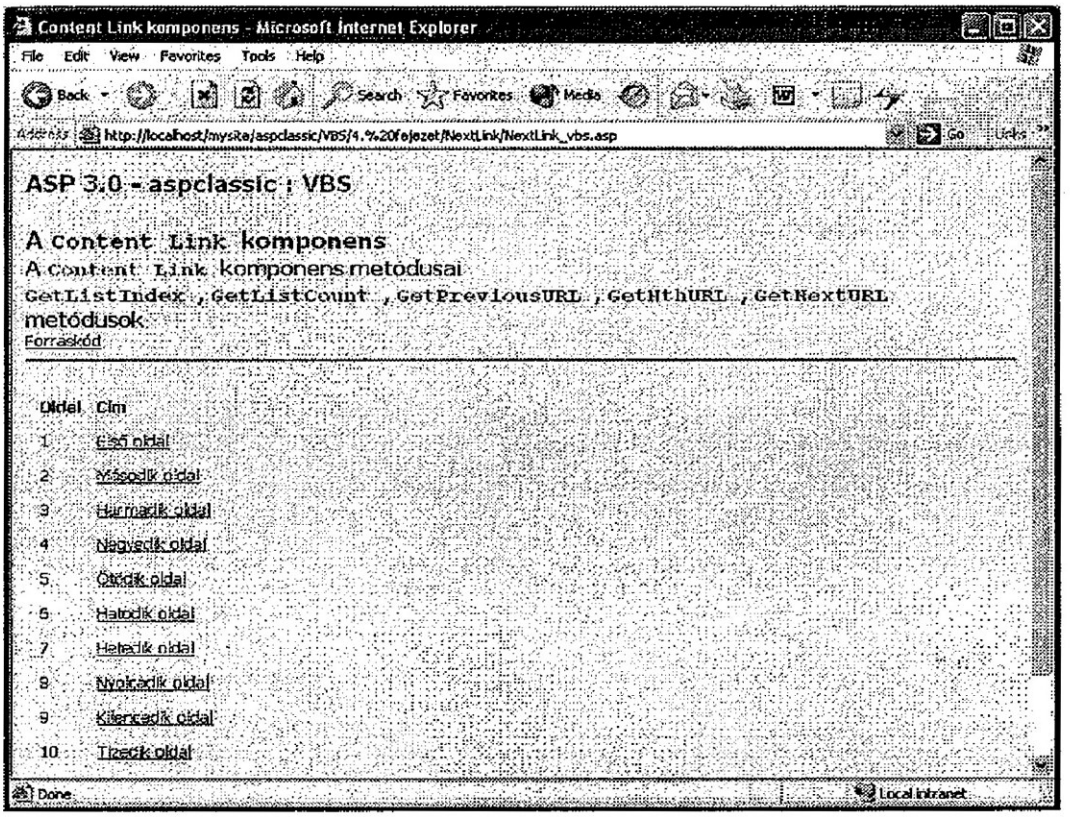
A `GetNthURL` és `GetNthDescription` metódusok a tartalomjegyzék elérési útját megadó paraméter mellett, egy további paraméterrel a tartalomjegyzék egy lapjának sorszámát várják és a megjelölt lap adatait adják vissza:

```
Dim strURL
Dim strDesc
strURL = objNextLink.GetNthURL(tocPath, n)
strDesc = objNextLink.GetNthDescription(tocPath, n)
```

A `NextLink` komponens egyetlen `About` tulajdonsága a komponens verziójáról nyújt információt és természetesen csak olvasható:

```
Dim strVersion
strVersion = objNextLink.About
```

A `NextLink` komponens használatát szemléltető példánk két állományból áll. Az első ASP lap, `NextLink/NextLink_vbs.asp` a `NextLink/toc.txt` tartalomjegyzék segítségével megjeleníti a 4.6. ábrán látható ASP tartalomjegyzéket, mely kapcsolatokat tartalmaz a `Pages` mappába elhelyezett lapokhoz.



4.6. ábra A `NextLink` *komponenssel előállított tartalomjegyzék*

A `Pages` mappában elhelyezett lapok egy `#include` utasítással beágyazzák a szintén ebben a mappában található `Navigation_vbs.asp` lapot. Ez az ASP lap egy újabb `NextLink` komponenst hoz létre, az információkat ezúttal a `nav.txt` szövegállományból véve (mely légyegében ugyanazt tartalmazza mint a `toc.txt` állomány, csak a lapok leírása hiányzik, mert itt nincs rá szükség).

Ha a 4.6. ábrán látható lap egyik kapcsolatára, például a *Negyedik oldal*-hoz vezető kapcsolatra kattintunk, akkor a megfelelő lap megjelenítésével a beágyazott `Navigation_vbs.asp` kódja is értelmezve lesz és megfelelőképpen alakul a lap megjelenítése.

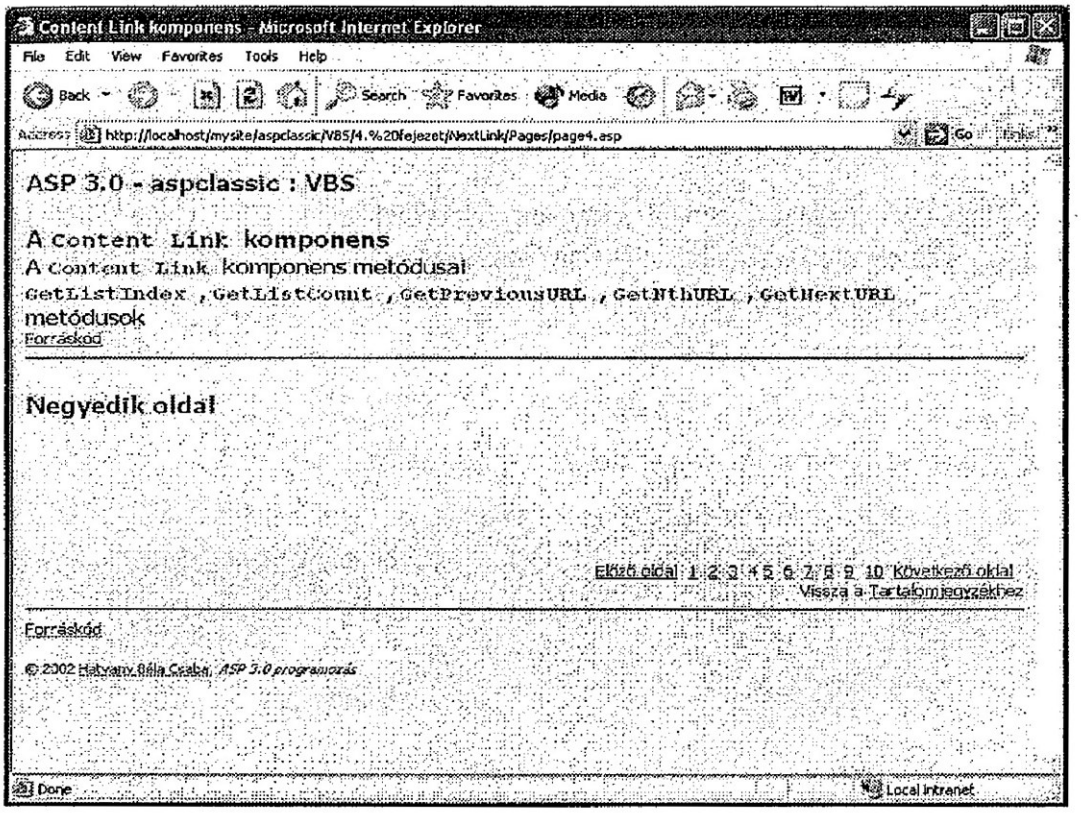
Nézzük meg a `Navigation_vbs.asp` lap forráskódját:

```
<%@ LANGUAGE="VBScript" CODEPAGE="1250" %>
<%

  Dim objNextLink
  Dim intListIndex
```

```
Dim intListCount
Dim strNavigationFile
Dim iCount
Dim p
Dim n

Set objNextLink = Server.CreateObject("MSWC.NextLink")
strNavigationFile = "nav.txt"
intListIndex = objNextLink.GetListIndex(strNavigationFile)
intListCount = objNextLink.GetListCount(strNavigationFile)

Response.Write "<div align=right>"
If intListIndex >1 Then
      p = intListIndex-1
      Response.Write "<a href=""page" & p & _
      ".asp"">Elõzõ oldal</a>  "
End If

For iCount=1 To intListCount
      If iCount = intListIndex Then
            Response.Write iCount & " "
      Else
            Response.Write "<a href=""page" & _
            iCount & ".asp"">" & iCount & "</a>  "
      End If
Next

If intListIndex < intListCount Then
      n = intListIndex+1
      Response.Write "<a href=""page" & n & _
      ".asp"">Következõ oldal</a>  "
End If
      Response.Write "<br>Vissza a " & _
      "<a href=""../NextLink_vbs.asp"">" & _
      "Tartalomjegyzék</a>hez"
      Response.Write "</div>"
%>
```

A forráskódban foglaltakat röviden így írhatjuk le. Minden egyes lapon megjelenítjük a tartalomjegyzékben szereplő összes lap sorszámát. Ha nem az aktuális lapról van szó, akkor a sorszámot egy kapcsolattal kötjük a megfelelő laphoz. Ha nem az első lap az aktuális lap, akkor megjelenítjük az **Előző** szöveget, amihez egy kapcsolattal az aktuális lap előzőjét kapcsoljuk. Ha nem az utolsó lap az aktuális lap, akkor megjelenítjük az **Következő** szöveget, amihez egy kapcsolattal az aktuális lap követőjét kapcsoljuk. Így egy olyan navigációs estközt hozunk létre, mint amilyent például a www.google.com keresőgép lapjain látunk.

Az eredményt a 4.7. ábrán láthatjuk.



4.7. ábra A `NextLink` *komponenssel előállított navigációs eszköz*

4.5 A `Counters` (számlálókat kezelő) komponens

A `Counters` komponenssel legelőször a második fejezetben találkoztunk. Akkor csak egy alkalmazásszintű objektum példájaként használtuk az alkalmazásunk `global.asa` állományában. Most közelebbről fogjuk ezt a komponenst megvizsgálni.

A `Counters` komponens számlálókat bocsát a rendelkezésünkre. Ilyen számlálók értelemszerűen alkalmazásszintűek legyenek, ezért leggyakrabban az alkalmazás `global.asa` állományában definiáljuk a komponens objektumát.

Ez a komponenst is, ugyanúgy mint a többi MSWC komponenst a `Server.CreateObject` metódussal:

```
Dim objCounters
Set objCounters = Server.CreateObject("MSWC.Counters")
```

vagy az `<OBJECT>` HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.Counters"
ID=" objCounters ">
</OBJECT>
```

hozhatjuk létre.

A Counters komponens DLL könyvtára a counters.dll állomány, mely rendszerint a winnt\system32\inetsrv mappában található. Ugyanebben a mappában helyezkedik el a data mappa és ezen belül a counters.txt szövegállomány.

Amint nemsokára látni fogjuk, a Counters komponens alkalmazásszintű változókat hoz létre és lát el értékkel. Ezeket a változókat számlálóknak nevezzük. A komponens a counters.txt szövegállományba könyveli el a számlálókat. A counters.txt állomány tipikus tartalma például ez lehet:

```
counter1:3
counter2:5
```

Minden egyes sor *countername*:*value* alakú, ahol *countername* a változó, azaz a számláló neve és *value* az aktuális értéke.

A Counters komponensnek négy metódusa van: Set, Get, Increment, Remove.

A Set metódussal egy számlálót definiálhatunk és elláthatjuk egy kezdőértékkel:

```
objCounters.Set countername, value
```

ahol *countername* a változó nevét tartalmazó karakterlánc és *value* egy **Long** típusú érték.

A Get metódussal egy megadott számláló értékét olvassuk ki:

```
Dim lngValue
lngValue = objCounters.Get(countername)
```

ahol *countername* a számláló neve. Ha a *countername* számláló nem létezik, akkor a metódus létrehozza és 0 értékkel látja el.

Az Increment metódus megemeli eggyel a számláló értékét és az új értéket adja vissza:

```
Dim lngValue
lngValue = objCounters.Increment(countername)
```

ahol *countername* a számláló neve. Ha a *countername* számláló nem létezik, akkor a metódus létrehozza és 1 értékkel látja el.

A Remove metódus eltávolítja a megnevezett számlálót a Counters komponensből:

```
objCounters.Remove(countername)
```

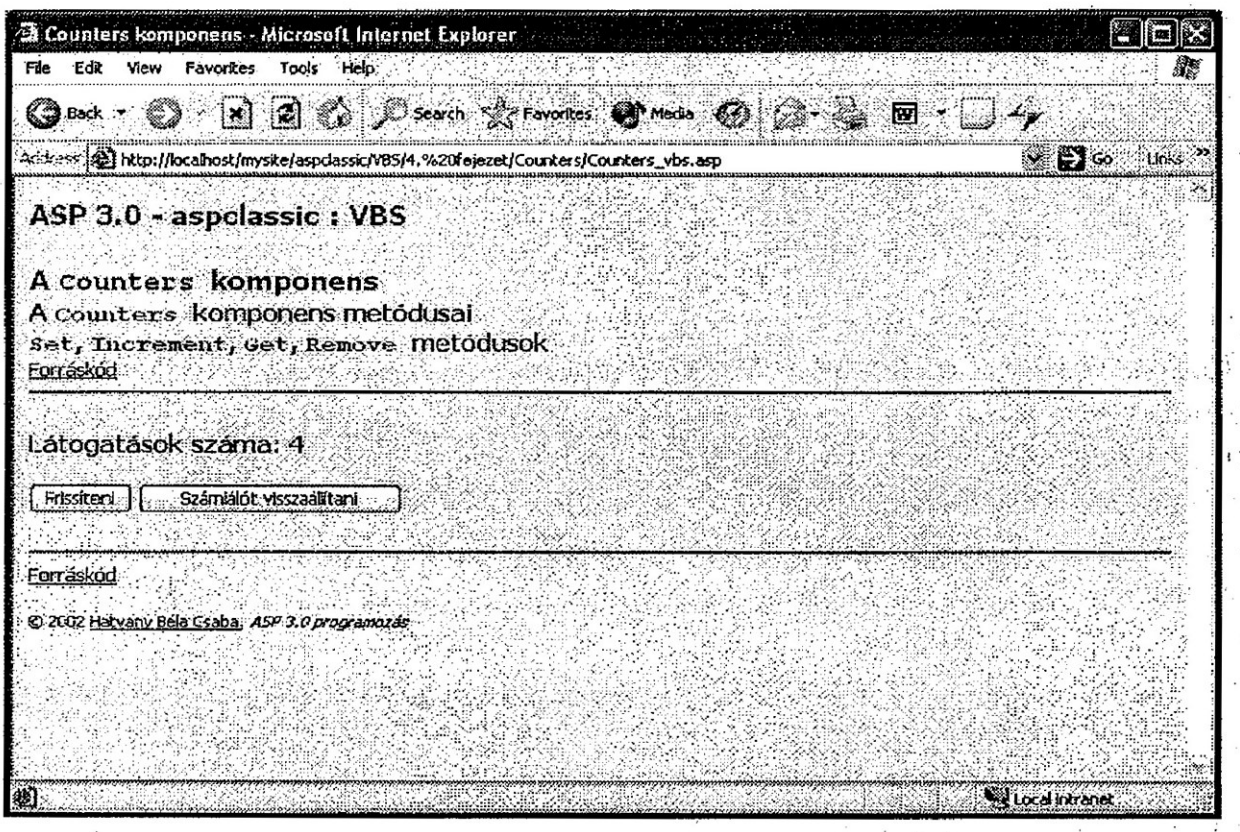
ahol *countername* a számláló neve. Egy számláló eltávolításával a rá vonatkozó bejegyzés ki lesz törölve a `counters.txt` állományból.

A `Counters` komponens alkalmazásának bemutatására felhasználjuk a VBS alkalmazásunk `global.asa` állományában létrehozott `myAppCounter` objektumot, mely a `Counters` komponens egy példánya. A `global.asa` állományban levő `Application_OnStart` eljárást bővítsük ki a következő utasítással:

```
myAppCounter.Set "Visits_VBS",0
```

Ezzel létrehozzuk a `"Visits_VBS"` nevű számlálót és ellátjuk `0` értékkel. A letöltött ZIP állományban a 4. fejezetnek megfelelő mappában található a `Counters` mappa. Ide helyeztem el a `Counters_vbs.asp` példaalkalmazást, amivel a `global.asa` állományban létrehozott számlálót módosíthatjuk. A példaalkalmazás kiírja a számláló aktuális értékét és a lap frissítésénél megemeli a számláló értékét. A lapon két parancsgomb van: az egyikkel frissíthetjük a lapot, a másikkal visszaállíthatjuk a számlálót.

A példaalkalmazást a 4.8. ábrán láthatjuk.



4.8. ábra A `Counters` *komponens alkalmazása*

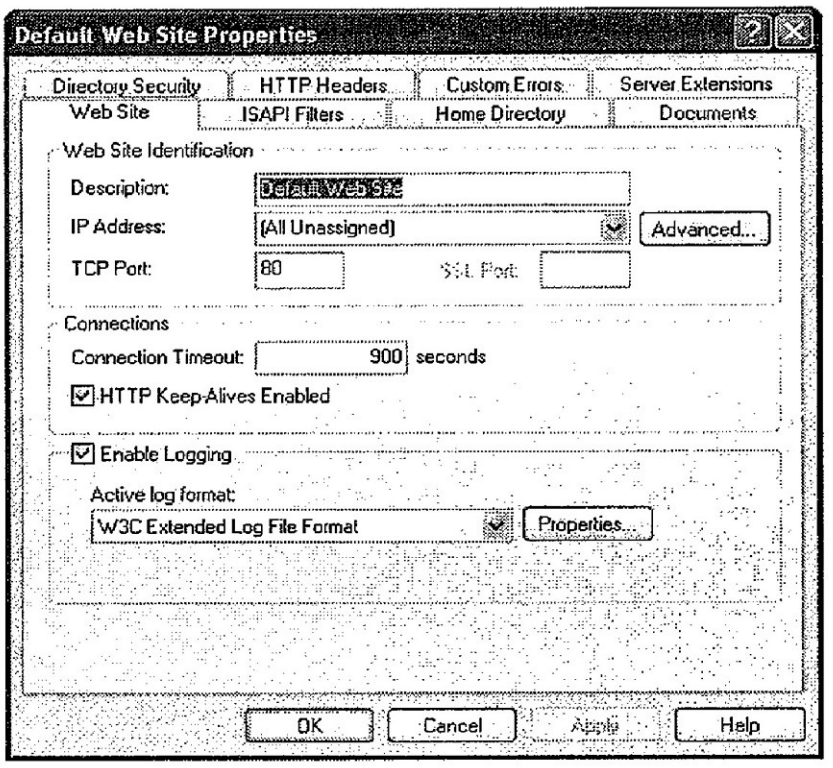
4.6 A IISLOG (naplókezelő) komponens

Web szerverek naplót (*logfile*) vezethetnek a szerveren elhelyezett webhelyekről. A naplóban tárolt adatok fontos információt szolgáltathatnak például üzleti döntések előkészítésénél. A napló kiértékelésével felmérhetjük a webhely látogatottságát, információt szerezhetünk az egyes lapok látogatottságáról, a látogatók földrajzi elosztásáról, megtudhatjuk melyik időszakban tetőzik és melyikben legalacsonyabb a lapok nézettsége, felmérhetjük melyik lapot tekintik meg a látogatók leghosszabb ideig és még sok más fontos információt nyerhetünk. Ezért nagyon fontos egy olyan komponens, mely segítségével gyorsan kiolvashatjuk a naplóban tárolt információt. Egy ilyen komponens az `IISLog` naplókezelő komponens. A komponens csupán az adatok kiolvasásához nyújt jelentős segítséget. Az adatok kiértékelése egy további feladat, amivel itt nem foglalkozunk.

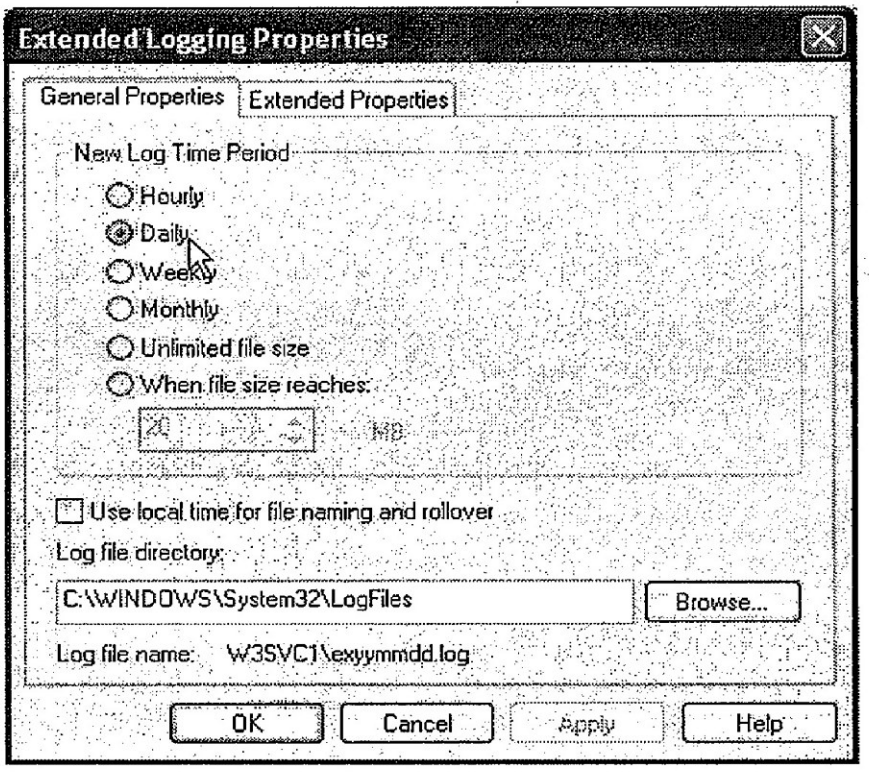
A naplóvezetést minden egyes webhely részére be- vagy kikapcsolhatjuk, illetve a napló tulajdonságait beállíthatjuk. Ha Windows 2000/XP Professional rendszerrel dolgozunk, akkor csak egy webhely áll rendelkezésünkre, ha viszont Windows 2000 Server-rel dolgozunk, akkor esetleg több webhely lehet a szerveren. Mindkét esetben a következőképpen kell eljárnunk, hogy a naplóvezetést bekapcsoljuk, illetve a napló tulajdonságait beállítsuk.

Az MMC konzolban nyissuk ki az IIS ablakát, kattintsunk az egérrel a webhelyünkre és nyomjuk meg az egér jobboldali gombját! A lenyíló környezetmenüben válasszuk ki a **Properties** menüpontot, hogy előhívjuk a **Default Web Site Properties** párbeszédablakot (4.9. ábra).

A naplóvezetést az **Enable Logging** jelölőnégyzet kijelölésével kapcsoljuk be. Az **Active log format** legördülő listából válasszuk ki a **W3C Extended Log File Format** formátumot, majd kattintsunk a **Properties...** gombra. Ezzel megjelenik az **Extended Logging Properties** című párbeszédablak (4.10. ábra). A **New Log Time Period** keretben válasszuk ki a **Dayly** opciót. Ezzel a beállítással a szerver minden nap egy új naplóállományt ír. A párbeszédablak alsó részén jelenik meg a naplóállomány neve: **Log file name: W3SVC1\exyymmdd.log**. Ezt a szöveget a következőképpen kell értelmezni. A **Log file directory** címke alatt alapértelmezésben, az egyébként megváltoztatható, mappa áll: `%WinDir%\System32\LogFiles`, ahol `%WinDir%` a rendszermappa. A naplóállományok a `%WinDir%\System32\LogFiles` mappában található `W3SVC1` mappában lesznek elhelyezve és nevük az **ex** előtagból, az év utolsó két számjegyéből (**yy**), a hónap kétszámjegyű sorszámából (**mm**) és a nap kétszámjegyű sorszámából (**dd**) áll.

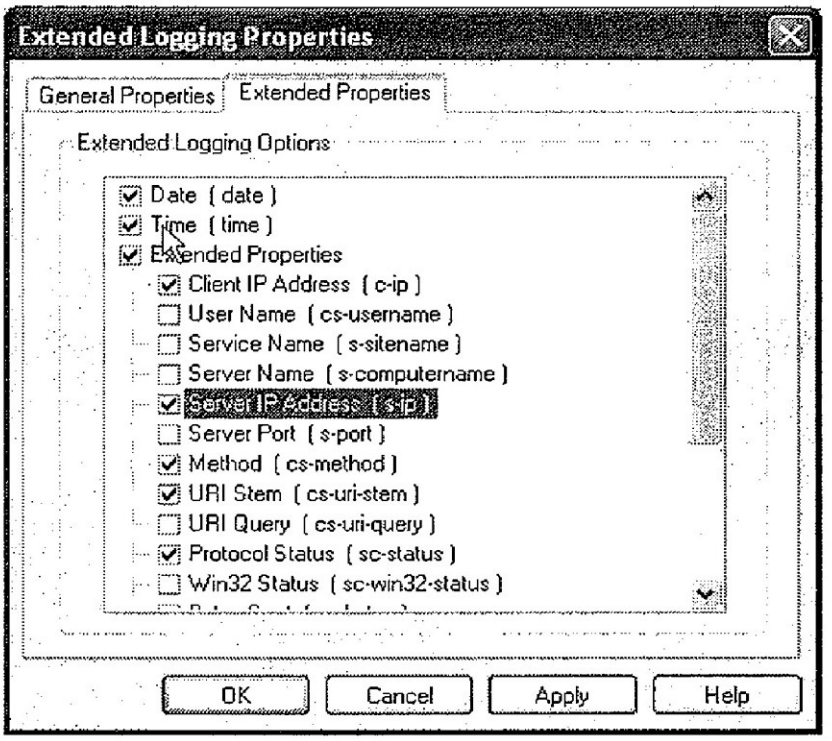


4.9. ábra A **Default Web Site Properties** *párbeszédablak*



4.10. ábra Az **Extended Logging Properties** *párbeszédablak*

Kattintsunk a párbeszédablakon az **Extended Properties** fülre! A megjelenő listában kijelölhetjük azokat az adatokat, amelyeket a naplóba be akarunk íratni (4.11. ábra).



4.11. ábra Kiválasztjuk a naplóba írandó adatokat

Válasszuk ki legalább a következő adatokat: dátum (*date*), időpont (*time*), a kliens IP-címe (*c-ip*) és a cél URL (*cs-uri-stem*). Ezzel a napló beállítását befejeztük.

Ha egy pár lapot letöltünk böngészőnkkel erről a webhelytől és ezután megnézzük a naplót, akkor látni fogjuk, hogy a szerver beírta a naplóba a kiválasztott adatokat. Minden egyes lapletöltésnek a napló egy sora felel meg, a sorban az adatokat egy szóköz választja el. Időnként a szerver megjegyzéseket is ír a naplóba, amit a sor elején egy # karakterrel vezet be.

Az `IISLog` komponens ilyen napló olvasását, ritkább esetben előállítását, segíti elő. Ezt a komponenst is a `Server.CreateObject` metódussal:

```
Dim objIISLog
Set objIISLog = Server.CreateObject("MSWC.IISLog")
```

vagy az `<OBJECT>` HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.IISLog"
ID=" objIISLog ">
</OBJECT>
```

hozhatjuk létre. A komponens DLL-je a `winnt\system32\intsev` mappában elhelyezett `iislog.dll` állomány.

A komponens a következő metódusokkal rendelkezik:

- `OpenLogFile`
- `ReadLogRecord`
- `WriteLogRecord`
- `ReadFilter`
- `AtEndOfLog`
- `CloseLogFile`

Az `OpenLogFile` segítségével megnyitjuk a naplóállományt. Szintaxisa a következő:

```
objIISLog.OpenLogFile(filename, iomode, _
servicename,serviceinstance,outputformat)
```

ahol

- *`filename`* a napló fizikai elérési útja;
- *`iomode`* két értéket kaphat: `ForReading` (= 1) alapértelmezett értéket, ha a naplót olvasásra nyitjuk és `ForWriteing` (= 2), ha a naplót írásra nyitjuk;
- *`servicename`* a szolgáltatás neve, alapértelmezésben `"W3SVC"`;
- *`serviceinstance`* a szolgáltatás példányszáma: az alapértelmezett webhely esetében 1;
- *`outputformat`* kimenet (írás) esetében egy a következő karakterláncok közül, mely meghatározza a kimenet formátumát: `"NCSA Common Log File Format"`, `"Microsoft IIS Log File Format"`, `"W3C Extended Log File Format"`. Olvasás esetében 0.

Fontos megjegyezni, hogy mindegyik paraméter kötelező.

A `ReadLogRecord` metódusnak nincs paramétere. Segítségével az olvasásra megnyitott naplóból a következő rekordot (sort) olvassuk ki:

```
objIISLog.ReadLogRecord
```

A rekord olvasása abban nyilvánul meg, hogy a soron levő rekord adataival megtölti az `objIISLog` objektum tulajdonságait, amiről lejjebb még szó esik.

A `WriteLogRecord` metódussal egy `IISLog` komponenssel olvasásra megnyitott naplóból kiolvasott rekordokat beírja egy másik állományba, melyet egy `IISLog` komponenssel írásra nyitottunk meg. A metódus egy paramétert vár, a napló olvasására használt `IISLog` objektumot. A következő kódrészlet megmutatja, miként alakítunk át egy `"W3C Extended Log File Format"` formátumú *`LogFile`* naplót egy `"W3C Extended Log File Format"` formátumú *`OutputFile`* naplóba:

```
Const ForReading = 1
Const ForWriting = 2
Dim objIISLog_Read
Dim objIISLog_Write
' létrehozunk egy IISLog objektumot
' az első napló olvasására
Set objIISLog_Read = CreateObject("MSWC.IISLog")
' létrehozunk egy IISLog objektumot
' a második napló írására
Set objIISLog_Write = CreateObject("MSWC.IISLog")
' megnyitjuk az első naplót olvasásra
objIISLog_Read.OpenLogFile LogFile, ForReading, _
"W3SVC", 1, "W3C Extended Log File Format"
' megnyitjuk a második naplót írásra
objIISLog_Write.OpenLogFile OutputFile, ForWriting, _
"W3SVC", 1, "NCSA Common Log File Format"

' beolvassuk az első napló első rekordját
objIISLog_Read.ReadLogRecord
' és beírjuk a második naplóba
' a formátum átalakításáról a komponens gondoskodik
objIISLog_Write.WriteLogRecord objIISLog_Read
' megismételjük ezt a műveletet ameddig nem
' olvastuk ki az első napló összes rekordját
Do While Not objIISLog_Read.AtEndOfLog
        objIISLog_Read.ReadLogRecord
        objIISLog_Write.WriteLogRecord objIISLog_Read
Loop
' zárjuk a naplókat
oRead.CloseLogFiles ForReading
oWrite.CloseLogFiles ForWriting
```

A példa a hátramaradt két metódus alkalmazását is bemutatja. Az `AtEndOfLog` mindig egy **`Boolean`** típusú értéket ad vissza: az állomány végén **`True`** értéket, különben **`False`** értéket.

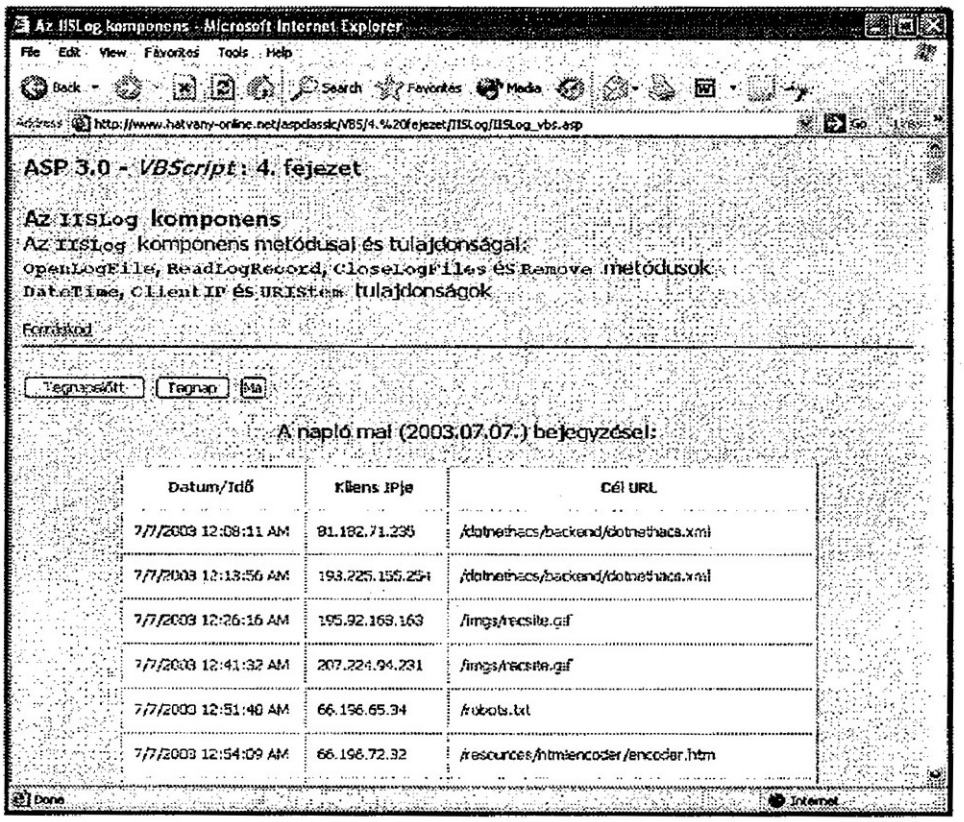
A `CloseLogFiles` metódusnak egy opcionális paramétere van, mely a `ForReading` vagy `ForWriting` értékeket veheti fel.

Az `IISLog` komponensnek húsz csak olvasható tulajdonsága van, melyek a naplóba írt rekordok adatmezőivel egyeznek meg. Az alábbi táblázatba összefoglaltam ezeket a tulajdonságokat:

Tulajdonság	Leírás
`BytesReceived`	Fogadott bájtok
`BytesSent`	Küldött bájtok
`ClientIP`	A kliens IP címe
`Cookie`	A kliens sütijei

Tulajdonság	Leírás
`CustomFields`	Egyéni adatok, mely a lekérdezés fejlécében voltak
`DateTime`	Dátum és idő GMT rendszerben
`Method`	A lekérdezés metódusa
`ProtocolStatus`	A protokoll státusa
`ProtocolVersion`	A protokoll verziója
`Referer`	A hívó lap címe
`ServerIP`	A szerver IPcíme
`ServerName`	A szerver neve.
`ServerPort`	A szerver portszáma.
`ServiceName`	A szolgáltatás neve.
`TimeTaken`	A letöltés időtartama.
`URIQuery`	A lekérdezéshez hozzáfűzött lekérdezési láncparaméterei
`URIStem`	A cél URL-je
`UserAgent`	A lekérdező böngésző azonosítója
`UserName`	A lekérdező gépre bejelentkezett felhasználó azonosítója
`Win32Status`	A lekérdezés státuskódja

Az `IISLog` komponens használatát a `4. fejezet\IISLog\IISLog_vbs.asp` példaalkalmazás mutatja be. Az alkalmazás segítségével a mindennapi és az egy, illetve két nappal előtti naplóból olvashatunk adatokat. Az alkalmazást a 4.12. ábrán láthatjuk a böngészőben.



4.12. ábra Az `IISLog` komponens kiolvassa az adatokat a naplóból

4.7 A `MyInfo` (egyedi információt kezelő) komponens

A `MyInfo` komponens egy egyszerű módszert kínál arra, hogy az alkalmazás bármelyik lapjáról elérhető egyedi információt tároljunk. Legtöbbször az egész webhelyre vagy az egész alkalmazásra vonatkozó információt szoktunk a `MyInfo` komponenssel tárolni, illetve a látogatóval közölni.

A `MyInfo` komponens létrehozásáról ugyanaz elmondható, mint a többi MSWC komponensről: a komponenst a `Server.CreateObject` metódussal:

```
Dim objMyInfo
Set objMyInfo = Server.CreateObject("MSWC.MyInfo ")
```

vagy az `<OBJECT>` HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.MyInfo "
ID="objMyInfo">
</OBJECT>
```

hozhatjuk létre. A komponens DLL-je a `winnt\system32\inetsrv` mappában elhelyezett `myinfo.dll` állomány. A különbség a többi komponenssel szemben abban áll, hogy a `MyInfo` komponens csak egy példányát hozzuk létre alkalmazásként vagy akár az egész webhely részére. Ezért rendszerint a komponenst megtestesítő objektumot az alkalmazás `global.asa` állományában definiáljuk például egy `<OBJECT>` HTML elem segítségéve.

A komponens nem rendelkezik előre megadott metódusokkal vagy tulajdonságokkal, viszont dinamikusan definiálhatunk tulajdonságokat és elláthatjuk értékkel, illetve lekérdezhetjük a megadott tulajdonságok értékét:

```
Dim myvalue
objMyInfo.property = value
myvalue = objMyInfo.property
```

Valahányszor a látogató elhagyja vagy frissíti a lapot, amelyen a `MyInfo` komponens egyik tulajdonságát megadjuk, a szerver beírja a komponens tulajdonságainak értékét a `winnt\system32\inetsrv\data` mappába elhelyezett `myinfo.xml` állományba. Ennek az állománynak egy különleges XML-szerű formátuma van:

```
<XML>
<property1>value1</>
<property2>value2</>
...
</XML>
```

Nézzük meg a 4. fejezet `MyInfo` mappájában található példaalkalmazást! Mielőtt az alkalmazást letöltenénk a böngészőnkbe, végezzük el a következő kiegészítéseket az alkalmazás `/aspclassic/VBS/global.asa` állományában. Ha az állományban nincs definiálva a `MyInfo` komponens egy példánya, akkor egészítsük ki az állományt a következő `<OBJECT>` elemmel:

```
<OBJECT ID="objMyInfo"
        RUNAT="Server"
        SCOPE="Application"
        PROGID="MSWC.MyInfo">
</OBJECT>
```

Példaalkalmazásunk két állományból áll.

A `MyInfoSet_vbs.asp` lappal értékeket adunk a létrehozott `objMyInfo` objektum tulajdonságainak:

```
<%
 Dim strNow
 strNow=Now
 objMyInfo.Title="ASP 3.0 programoz&#225;s"
 objMyInfo.Author="Hatvany B&#233;la Csaba"
 objMyInfo.AuthorEmail="csaba@hatvany-online.net"
 objMyInfo.AuthorHomepage="www.hatvany-online.net"
 objMyInfo.Publisher="ComputerBooks.Kft"
 objMyInfo.PublisherEmail="info@computerbooks.hu"
 objMyInfo.PublisherHomepage="www.computerbooks.hu"
 objMyInfo.Time=strNow
 Response.Write strNow & ": Kiírtuk a tulajdonságokat.<br>"
 Response.Write "<A HREF=""MyInfo_vbs.asp"">Megnézni</A>"
%>
```

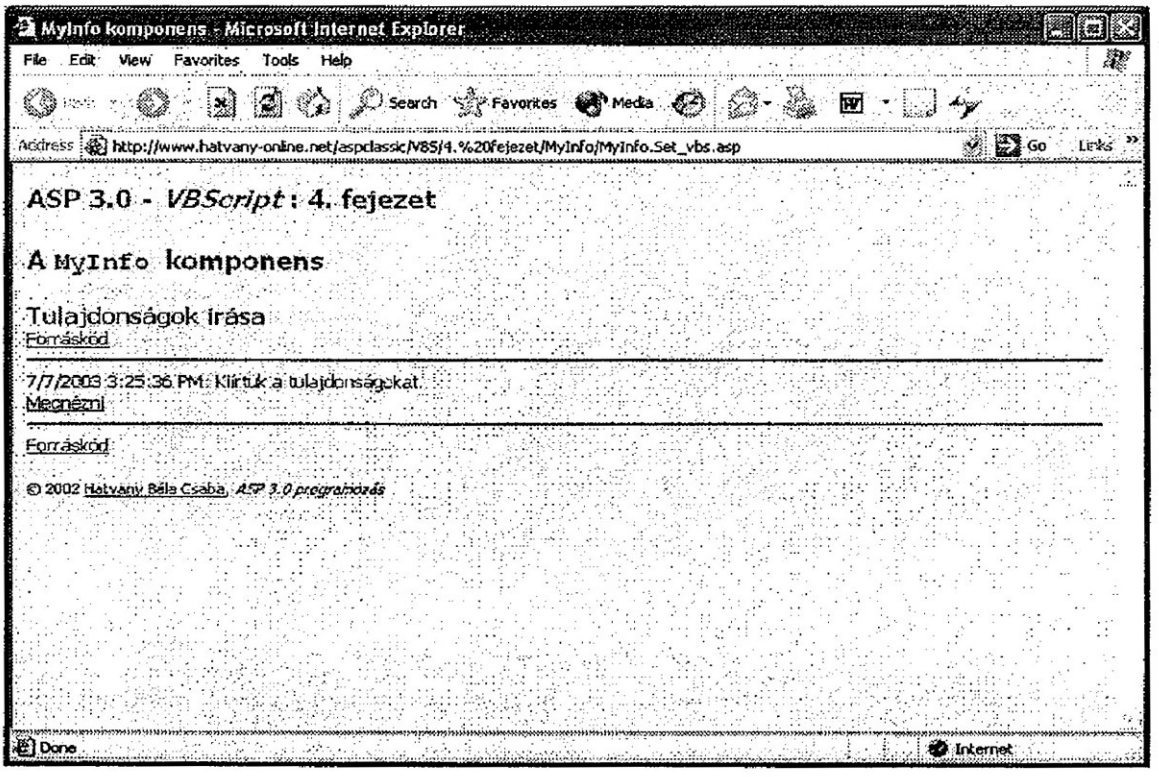
A tulajdonságok információt tartalmaznak erről a könyvről, a könyv kiadójáról és szerzőjéről. A legutolsó tulajdonság a lap letöltésének időpontját rögzíti. Ezt az utolsó tulajdonságot csak azért használom, hogy egy kis kísérletet elvégezhessünk.

A második lap, `MyInfoGet_vbs.asp`, lekérdezi és megjeleníti az előbbi tulajdonságokat:

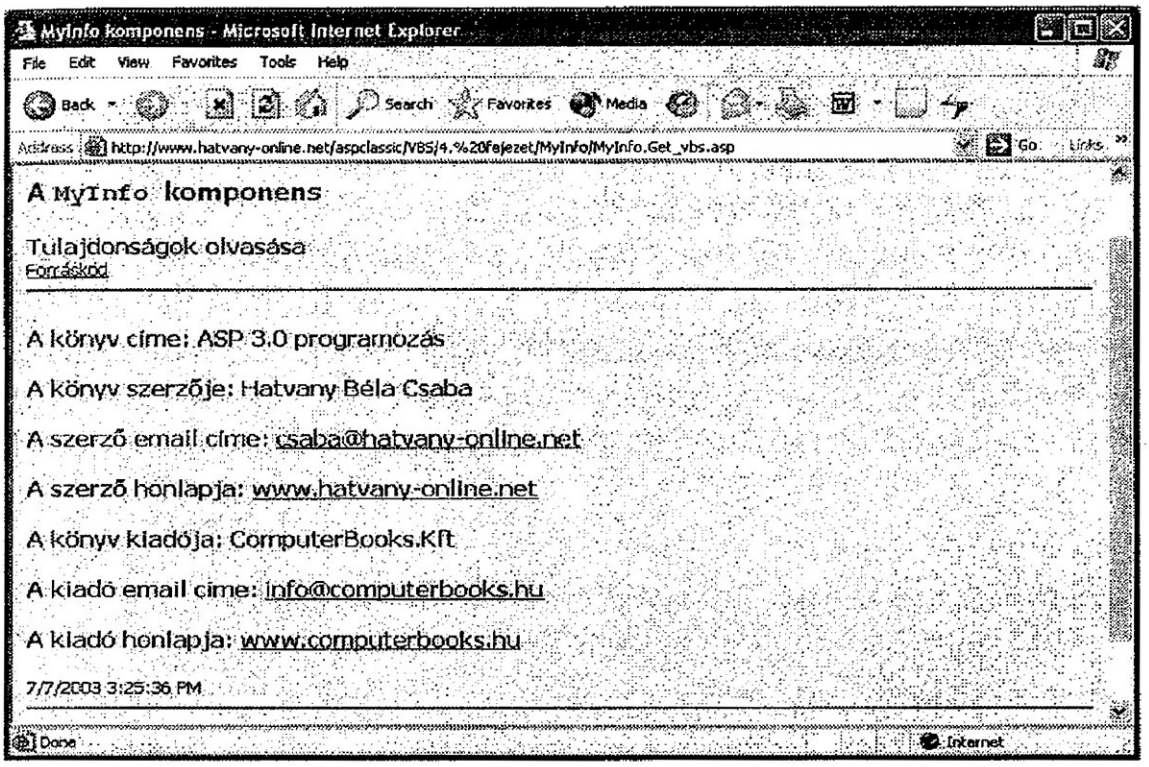
```
<%
 Response.Write "<H2><b>A könyv címe: </b>" & _
 objMyInfo.Title & "</H2>"
 Response.Write "<H2><b>A könyv szerzõje: </b>" & _
 objMyInfo.Author & "</H2>"
 Response.Write "<H2><b>A szerzõ email címe: " & _
 </b><A HREF=""mailto:" & _
 objMyInfo.AuthorEmail & """>" & _
 objMyInfo.AuthorEmail & "</A></H2>"
 Response.Write "<H2><b>A szerzõ honlapja: " & _
```

```
</b><A HREF=""http://" & _
objMyInfo.AuthorHomepage & """ target=_new>" & _
objMyInfo.AuthorHomepage & "</A></H2>"

Response.Write "<H2><b>A könyv kiadója: </b>" & _
objMyInfo.Publisher & "</H2>"
Response.Write "<H2><b>A kiadó  email címe: " & _
</b><A HREF=""mailto:" & _
objMyInfo.PublisherEmail & """>" & _
objMyInfo.PublisherEmail & "</A></H2>"
Response.Write "<H2><b>A kiadó honlapja: " & _
</b><A HREF=""http://" & _
objMyInfo.PublisherHomepage & """ target=_new>" & _
objMyInfo.PublisherHomepage & "</A></H2>"
Response.Write objMyInfo.Time
%>
```

Legelőször töltsük le a `MyInfoSet_vbs.asp` lapot. Jegyezzük meg a megjelenített időpontot. A 4.13. ábrán láthatjuk a lapot a böngészőben. A letöltés időpontja 16:28 volt. Most kattintsunk a **Megnézni** kapcsolatra. Ezzel az `objMyInfo` objektum tulajdonságait kiolvassuk és megjelenítjük. A 4.14. ábrán láthatjuk, hogy a megjelenített időpont szintén 16:28, ami elvárható volt, hiszen egy alkalmazásszintű objektum (`objMyInfo`, amit a `global.asa` lap hozott létre) tulajdonságát az egyik lapon (`MyInfoSet_vbs.asp`) értékkel ellátunk, egy másik lapon pedig (`MyInfoGet_vbs.asp`) leolvastuk.



4.13. ábra A `MyInfo` *komponens dinamikus tulajdonságainak értékeket adunk*



4.14. ábra Kiolvassuk a `MyInfo` *komponens dinamikus tulajdonságainak értéket*

Most állítsuk le a szerverünket. Ez legegyszerűbben úgy érjük el, hogy kikapcsoljuk a gépünket. Ezzel biztosak lehetünk, hogy a `objMyInfo` objektum eltűnt a memóriából. Indítsuk újra el a gépünket és az IIS szervert. Ezúttal ne a `MyInfoSet_vbs.asp` lapot, hanem a `MyInfoGet_vbs.asp` lapot töltsük le. Az alkalmazás egyik lapjának letöltésével a `global.asp` lap ugyan létrehozza az `objMyInfo` objektumot, de ha nem a `MyInfo` komponens egyik példányáról lenne szó, akkor ez az objektum nem rendelkezhet egy olyan tulajdonsággal, melynek értékét a gépünk leállítása előtt határoztuk meg. Nem így a `MyInfo` komponens esetében: a letöltött `MyInfoGet_vbs.asp` lapon feltüntetett időpont változatlanul 16:28. Honnan veszi az objektum ezt az értéket? Itt lép a színre a `myinfo.xml` állomány. Ha egy ASCII szövegszerkesztővel, például a **NotePad**-del, megnézzük a `myinfo.xml` állomány tartalmát, akkor a következőket találjuk:

```
<XML>
<Title>ASP 3.0 programoz&#225;s</>
<Author>Hatvany B&#233;la Csaba</>
<AuthorEmail>csaba@hatvany-online.net</>
<AuthorHomepage>www.hatvany-online.net</>
<Publisher>ComputerBooks.Kft</>
<PublisherEmail>info@computerbooks.hu</>
<PublisherHomepage>www.computerbooks.hu</>
<Time>31.01.2003 16:28:00</>
</XML>
```

(Az állományt nem nézhetjük meg az Internet Exporer-rel: mivelhogy az állomány nem egy szabályos XML állomány, a böngésző hibát jelent).

A kísérlet utolsó részében nézzük meg a `myinfp.xml` állományt a **TextPad** szerkesztővel vagy a **MS Visual InterDev** szerkesztőjével (sajnos a **NodePad** szerkesztő nem felel meg ebben az esetben) és hagyjuk a szerkesztő ablakát nyitva. Töltsük le újra a `MyInfoSet_vbs.asp` lapot (a biztonság kedvéért frissíthetjük is), majd hozzuk a szerkesztő ablakát az előtérbe. A szerkesztő jelenteni fogja, hogy egy másik alkalmazás módosítást eszközölt a `myinfo.xml` lapon, és megkérdezi nem kívánjuk-e az új tartalmát megtekinteni. Ez bizonyítja, hogy az `objMyInfo` objektum tulajdonságainak módosításakor a szerver napirendre hozza a `myinfo.xml` állományt.

4.8 A `PageCounter` (lapszámláló) komponens

A `PageCounter` komponens segítségével számon tarthatjuk, hány alkalommal lett egy adott lap megnézve. A komponens a `winnt\system32\inetsrv\data` mappában található `HitCnt.cnt` szövegállományba tárolja a látogatások számát. Az állományban minden egyes számon tartott lap számára a látogatások, vagy ahogy még mondani szokták, a lapot ért találatok (*hit*) számát és a lap URL-jét írja be a komponens:

```
n1 pageURL1
n2 pageURL2
...
```

Ezt a komponenst is ugyanúgy hozhatjuk létre, mint az eddigieket: a `Server.CreateObject` metódussal:

```
Dim objPageCounter
Set objPageCounter = Server.CreateObject("MSWC.PageCounter ")
```

vagy az `<OBJECT>` HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.PageCounter "
ID="objPageCounter">
</OBJECT>
```

A komponensnek három metódusa van: `Hits`, `PageHit` és `Reset`.

A `Hits` metódussal lekérdezhetjük hányszor lett megtekintve egy adott lap? A metódusnak egy opcionális paramétere van, a szóban forgó lap URL-je (a *domain*-név nélkül). A metódus a lapot ért találatok (*hit*) számát adja vissza:

```
Dim lngHits
lngHits = objPageCounter.Hits([pageURL])
```

Ha a paramétert elhagyjuk, akkor a metódus az aktuális lap találatait adja vissza.

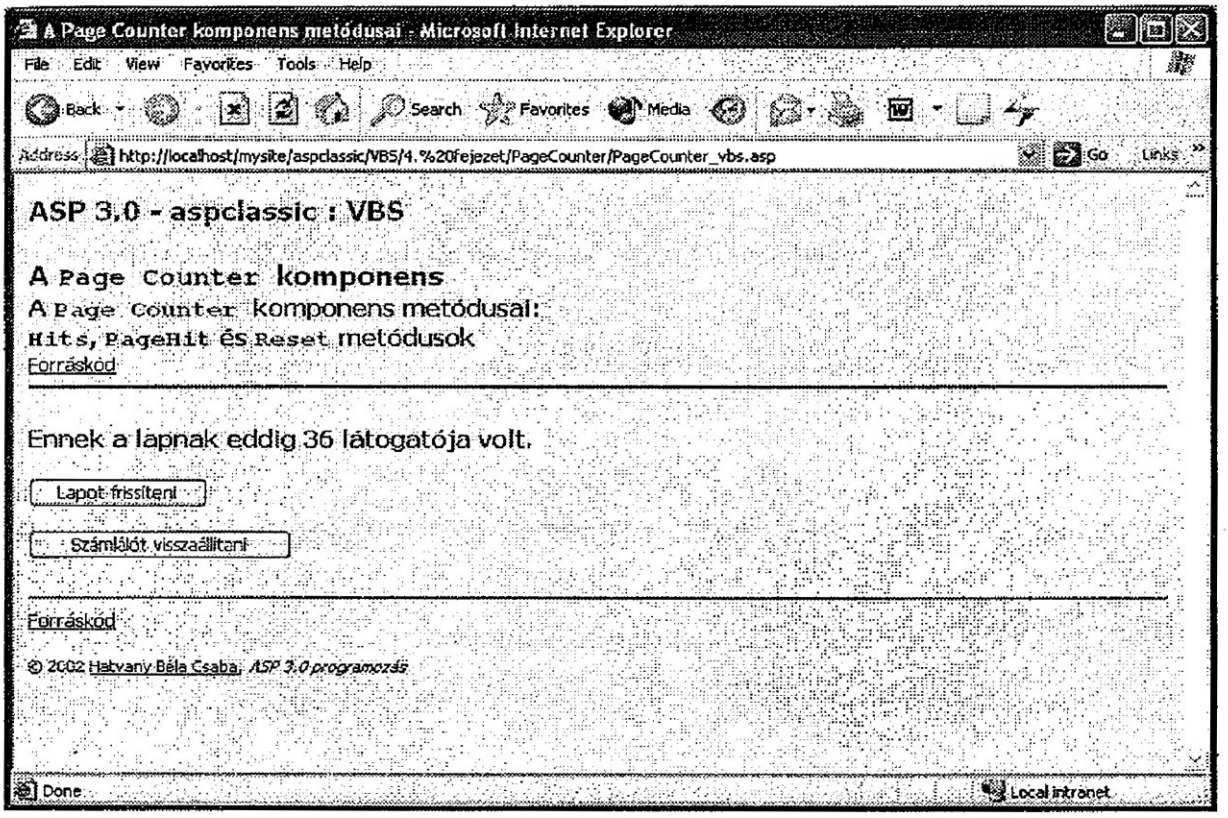
A `PageHit` metódus megemeli eggyel az aktuális lap találatainak számát. A metódusnak nincs paramétere és a találatok módosított értékét adja vissza:

```
Dim lngHits
lngHits = objPageCounter.PageHit
```

A `Remove` metódus visszaállítja `0`-ra a paraméterrel megadott lap találatainak számát. A paraméter opcionális és a szóban forgó lap URL-je legyen (*domain*-név nélkül). Ha a paramétert elhagyjuk, az aktuális lap találatainak számát állítja nullára:

```
objPageCounter.Remove([pageURL])
```

A `PageCounter` mappában található `PageCounter_vbs.asp` példalap ezt a komponenst alkalmazza a találatok számontartására. A 4.15. ábra a letöltött lapot mutatja.



4.15. ábra A laptalálatokat számláló `PageCounter` *komponens alkalmazása*

4.9 A PermissionChecker (jogosultság-ellenőrző) komponens

A PermissionChecker egyike a legegyszerűbb az MSWC komponensek közül. Segítségével megállapíthatjuk, hogy a látogató jogosult-e a megadott állományokat megnézni vagy erőforrásokat elérni.

A komponens DLL könyvtára a winnt\system32\inetsrv\PermChk.dll állomány. A komponens létrehozása a megszokott módon történik: a Server.CreateObject metódussal:

```
Dim objPermissionChecker
Set objPermissionChecker = _
Server.CreateObject("MSWC.PermissionChecker")
```

vagy az <OBJECT> HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.PermissionChecker"
ID="objPermissionChecker">
</OBJECT>
```

A komponensnek egy metódusa van, HasAccess. A metódus paramétere az ellenőrzött állomány virtuális vagy fizikális elérési útját adja meg, míg a komponens visszatérési értéke **True**, ha a látogató jogosult az állomány elérésére és **False**, ellenkező esetben:

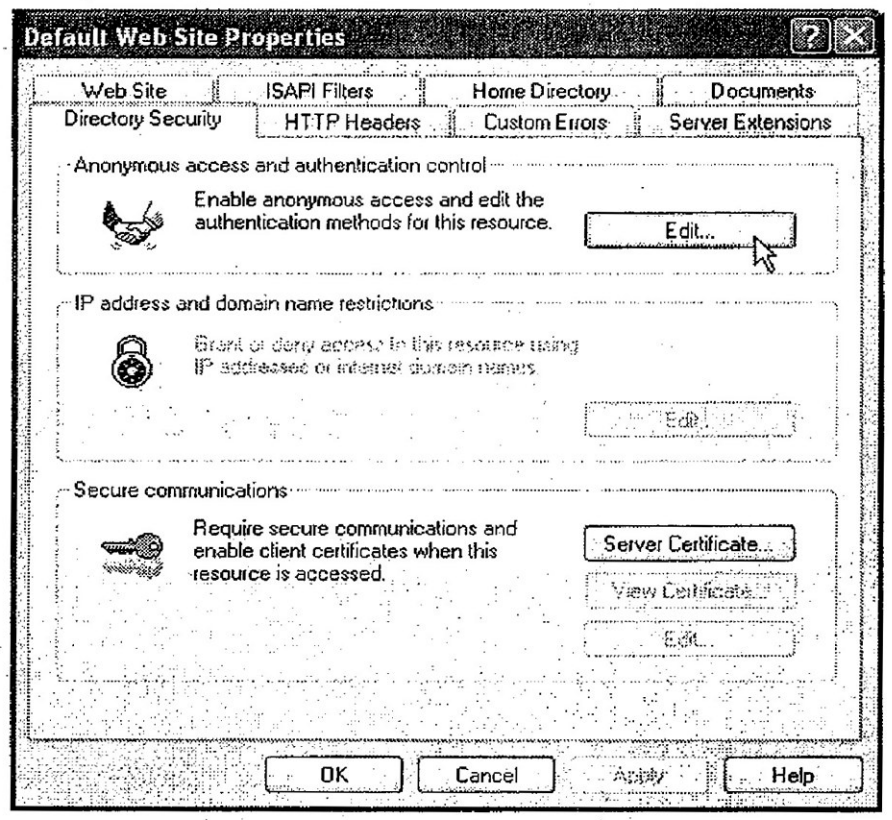
```
Dim blnAccess
blnAccess = objPermissionChecker(path)
```

Itt tisztáznunk kell egy-két fogalmat. Hogy állapítja meg a PremissionChecker komponens az aktuális látogató kilétét? Miszerint dönti el a komponens, kinek van és kinek nincs joga bizonyos erőforrásokhoz hozzáférni? Mielőtt továbbmennénk, tisztáznunk kell ezeket a kérdéseket.

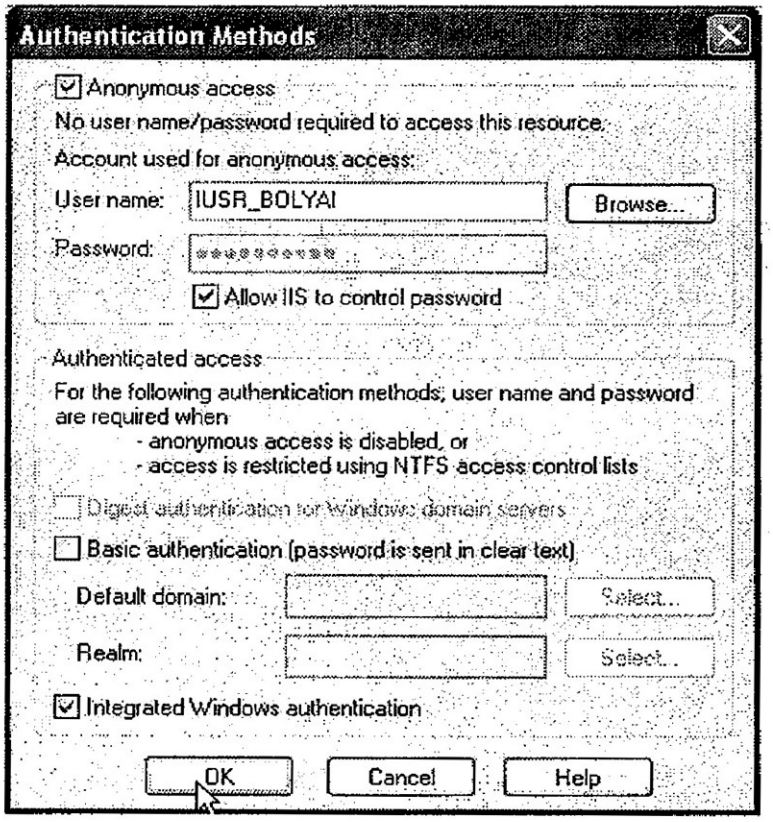
Az ASP alkalmazások leggyakrabban névtelen hozzáférést fogadnak el. Ez azt jelenti, hogy a látogatónak nem kell azonosítania magát, amikor egy lapot letölt. A szerver szintjén ilyenkor a hozzáférés az IIS azonosítójával, az IUSR_*machinename* azonosító felhasználásával történik, ahol *machinename* a szerver neve. A PermissionChecker komponens szempontjából ez azt jelenti, hogy ilyenkor az IUSR_*machinename* felhasználót hozzáengedi az alkalmazás lapjaihoz. Hogy állíthatjuk ezt az egyébként alapértelmezésű konfigurációt be?

Az IIS konzolban válasszuk ki azt a mappát vagy lapot melyet névtelen hozzáférésre akarjuk beállítani, nyomjuk le az egér jobb gombját és a lenyíló környezetmenüből

válasszuk ki a **Properties...** menüpontot. A megjelenő párbeszédablakon válasszuk ki a **Directory Security** fület (4.16. ábra) és nyomjuk meg az **Edit...** gombot.



4.16. ábra Egy virtuális mappa vagy lap biztonságának beállítása

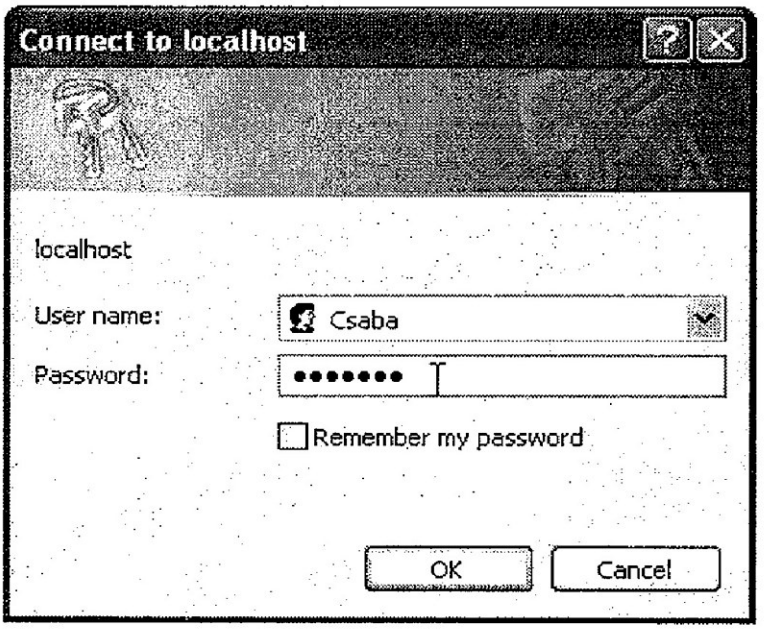


4.17. ábra A névtelen hozzáférés beállítása

A következő ablakon jelöljük ki az **Anonymous access** jelölőnégyzetet, hogy névtelen hozzáférést biztosítsunk, majd nyomjuk meg az **Edit...** gombot (4.17. ábra). A megjelenő ablak (4.18.) megmutatja az alapértelmezett IIS azonosítóját, amit névtelen hozzáférés esetében használ a szerver. Mint tudjuk ez az azonosító `IUSR_machinename` formátumú. Az ábrán láthatjuk, hogy az én gépemen ez a név `IUSR_BOLYAI` (rendszeremet 2002-ben vettem, Bolyai János születésének 200. évfordulója évében és tiszteletére róla neveztem el a gépemet.)

Névtelen hozzáférés esetén a `PermissionChecker` komponens az alapértelmezett `IUSR_machinename` azonosítót ellenőrzi. Ebben az esetben IIS saját azonosítójával, azaz az `IUSR_machinename` azonosító felhasználásával próbál a szóban forgó erőforráshoz hozzáférni, tehát a `PrermissionChecker` komponens `HasAccess` metódusa a **`True`** értéket adja vissza.

Bizonyos esetekben azonban, leggyakrabban intranetes alkalmazásokban kívánatos, hogy bizonyos lapok hozzáférésére csak egyes látogatók legyenek jogosultak. Ilyen esetekben a szóban forgó lapok hozzáférési beállításait módosítani kell. Ezt a 4.17. ábrán látható ablakon végezzük el. Ügyeljünk arra, hogy az **Anonymous access** jelölőnégyzet ne legyen kijelölve és jelöljük ki a **Basic authentication (password is send in clear text)** jelölőnégyzetet. Ezzel elérjük, hogy egy nem névtelen hozzáférésű lap letöltése előtt a böngésző egy párbeszédablakkal kéri a látogató azonosítását (4.18. ábra).



4.18. ábra A nem névtelen hozzáférésű lapok letöltésénél a látogatónak azonosítania kell magát

Ilyenkor a `PermissionChecker` komponens a lekérdező böngészőtől kapja (nem rejtjelezett formában) a bejelentkező látogató felhasználónevét és jelszavát. Ezeket az

adatokat hasonlítja a komponens össze az állományt megtekinteni jogosultak adataival. (Egy állomány vagy mappa hozzáférésére jogosult felhasználókat a **Windows Exporerben** a **Properties...** menüponttal adhatjuk meg.)

A `PermissionChecker` komponens alkalmazását a 4. fejezet mappájában található `PermissionChecker/PermissionChecker_vbs.asp` alkalmazás illusztrálja.

4.10 A `Tools` (eszköztár) komponens

A `Tools` komponens az utolsó az MSWC komponensek közül, amelyiket még meg kell ismerjünk. Segítségével egy pár hasznos eszközhöz jutunk: ellenőrizhetjük, hogy létezik-e egy állomány, dinamikusan létrehozhatunk ASP lapokat egy előre megadott minta vagy sablon szerint és véletlenszerűen egész számokat generálhatunk.

A metódus DLL-je a `winnt\system32\inetsrv` mappába elhelyezett `tools.dll` állomány. A komponenst az ismert két módszerrel hozhatjuk létre: a `Server.CreateObject` metódussal:

```
Dim objTools
Set objTools = Server.CreateObject("MSWC.Tools")
```

vagy az <OBJECT> HTML elem segítségével:

```
<OBJECT RUNAT="SERVER" PROGID="MSWC.Tools"
ID=" objTools ">
</OBJECT>
```

A komponens három metódussal rendelkezik: `FileExists`, `ProcessForm` és `Random`.

A `FileExists` metódus szintaxisa

```
Dim blnExists
blnExists = objTools.FileExists(fileURL)
```

ahol *`fileURL`* az ellenőrzött állomány virtuális elérési útja (*domain*-név nélkül). Ha a megnevezett állomány létezik, akkor a metódus **`True`** értéket ad vissza, különben **`False`** értéket.

A `FileExists` metódust a `Tools/Tools.FileExists_vbs.asp` mutatja be.

A `ProcessForm` metódus segítségével dinamikusan generálhatunk ASP lapokat. A létrehozandó lapokat egy előre megadott mintalap vagy sablon írja le. A mintalap egy

szabályszerű ASP lap azzal a kiegészítéssel, hogy <%% és %%> között dinamikusan változó ASP utasításokat tartalmazhat. A komponens egy úgynevezett alaplapot is használhat, melynek egyes részeit átveheti a generált lapba.

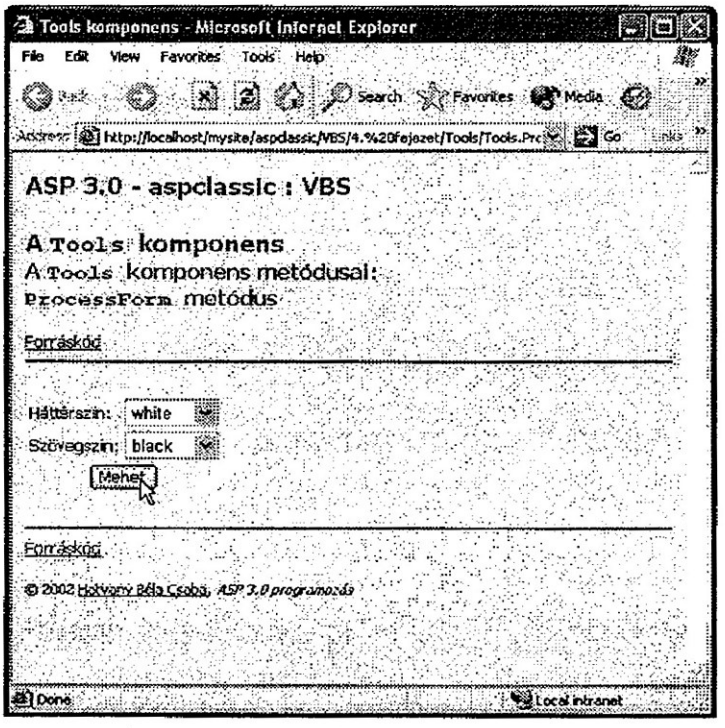
A metódus szintaxisa a következő:

```
objTools.ProcessForm(base, template, ipoint)
```

ahol

- `base` az alaplap URL-je (*domain*-név nélkül). Ez az állomány akkor is kell hogy létezzen, ha semmit sem veszünk át belőle és teljesen felül lesz írva;
- `template` a mintalap URL-je (*domain*-név nélkül);
- `ipoint` opcionális paraméter, beillesztési vagy beszúrási pont (*insertion point*). Ha megadjuk, akkor egy, a `base` alaplapban szereplő, karakterláncot ad meg. Ha a paraméterben ezt a karakterláncot egy * előzi meg, akkor a lap dinamikusan generált része az alaplapba a paraméterrel megadott karakterlánc első megjelenése elé lesz beleszúrva, különben utána lesz beleszúrva. Ha a paramétert teljesen elhagyjuk, akkor az alaplap teljes tartalma felül lesz írva.

A `Tools` mappába elhelyezett `Tools.ProcessForm_vbs.asp` alkalmazás bemutatja a `ProcessForm` használatát. Az alkalmazás kezdőlapján egy egyszerű <FORM> elem és két <SELECT> elem segítségével kiválaszthatjuk a generálandó lap háttér- és szövegszínét (4.19. ábra).



4.19. ábra A `ProcessForm` *metódus alkalmazása*

A `<FORM>` elem `ACTION` attribútuma egy másik ASP állományra utal, a `Tools.ProcessForm` mappában található `ProcessForm_vbs.asp` állományra. Ezen a lapon belül hozzuk létre a `Tools` komponens egy példányát és hívjuk meg a `ProcessForm` metódusát:

```
<%
 Dim objTools
 Dim l
 Dim dir

 Set objTools = Server.CreateObject("MSWC.Tools")
 l=InStrRev(Request.ServerVariables("SCRIPT_NAME"), "/")
 dir=Mid(Request.ServerVariables("SCRIPT_NAME"),1,l)
 objTools.ProcessForm dir & "base.asp", dir & "template.asp"
 Server.Execute "base.asp"
%>
```

A paraméterként megadott `base.asp` állományt üresen hagyjuk, a `template.asp` állomány tartalma pedig a következő:

```
<%@ LANGUAGE="VBScript" CODEPAGE="1250" %>
<HTML>
<HEAD>
<TITLE>Tools.ProcessForm metódus</TITLE>
</HEAD>
<BODY BGCOLOR=<%%= Request.Form("bgcolor") %%> TEXT=<%%=
Request.Form("text")%%>>
<H1>Ezt a lapot a <font face="Courier New">Tools</font>
komponenssel generáltuk!</H1>
<h2>Háttérszín: <%%= Request.Form("bgcolor")%%></h2>
<h2>Szövegszín: <%%= Request.Form("text")%%></h2>
</BODY>
</HTML>
```

A `<%%` és `%%>` közötti ASP kódrészletet a lap dinamikusan tölti ki. Ha például a kezdőlapon háttérszínnek zöldet (*green*) és szövegszínnek sárgát (*yellow*) választunk ki, akkor a komponens az alaplapba a mintalap alapján a következőket írja:

```
<%@ LANGUAGE='JScript' CODEPAGE='1250' %>
<HTML>
<HEAD>
<TITLE>Tools.ProcessForm metódus</TITLE>
</HEAD>
<BODY BGCOLOR=green TEXT=yellow>
<H1>Ezt a lapot a <font face="Courier New"> Tools</font>
komponenssel generáltuk!</H1>
<h2>Háttérszín: green</h2>
<h2>Szövegszín: yellow</h2>
</BODY>
</HTML>
```

A `Tools` komponens `Random` metódusa nem rendelkezik paraméterrel és minden meghívásakor egy véletlenszerűen generált, –32768 és 32767 közötti, egész számot ad vissza:

```
Dim intNumber
intNumber = objTools.Random
```

A `Random` metódust alkalmazva a `Tools/Tools.Random_vbs.asp` példaalkalmazás véletlenszerű sorrendben öt latin aforizmát jelenít meg.

4.11 A CDONTS email-komponens

A jelen fejezet utolsó két szakaszában az email üzenetek szerkesztésével és elküldésével foglalkozunk. Az email-ek elküldésére szerverkomponenst használunk. Erre a célra a szoftverpiacon sok komponens található, különböző árkategóriákban. Ugyanakkor az IIS telepítésével a Microsoft email-komponensei is gépünkre kerülnek és ezekkel is ugyanolyan könnyen és biztonságosan küldhetünk email-eket, mint a nemegyszer igen magas árú szoftveripari komponensekkel.

Alapjában véve két email-komponensről van szó: a CDONTS és a CDOSYS szerverkomponensekről. A CDO betűszó a *Collaboration Data Object* kifejezést rejti, amit magyarul adat-közreműködési vagy adat-együttműködési objektumot jelent. Ezt a két komponenst, illetve objektumot kollaborációs komponensnek, illetve objektumnak is szoktuk nevezni.

Mindkét komponens egy komplex objektummodellel rendelkezik, mely ugyanolyan összetett feladatok megoldását teszi lehetővé. Könyvünk tárgyát meghaladja ezen komponensek kimerítő leírása, az összes metódus és tulajdonság részletes tárgyalása. Ezért, mindkét komponens esetében csak a legszükségesebbet magyarázom el: miként lehet segítségükkel email-eket küldeni? Az érdeklődő Olvasó a letöltött ZIP állományban talál utalást a további információra.

A CDONTS (*CDO for NT systems*) komponens a régebbi és a **Windows NT** szerverek világához tartozik, CDOSYS viszont a **Windows 2000** szerveren található. Sok webszolgáltató azonban Windows 2000 szerverre is telepíti a CDONTS komponenst, hogy régebbi alkalmazások zökkenőmentes működését is biztosítsa.

Ebben a szakaszban a CDONTS komponenst használatát fogjuk megismerni.

Legegyszerűbb az lesz, ha egy függvénybe foglalom a CDONTS komponensről szükséges tudnivalókat és ennek kapcsán elmagyarázom miként működik.

Itt a függvény forráskódja:

```
<%
' elküld egy email-t a CDONTS segítségével
' paraméterek:
' strFrom - feladó
' strTo - címzett
' strSubject - tárgy
' strBody - üzenet törzse
' intFormat - üzenet formátuma:
' =0 - HTML, =1 - szöveg
Function SendCDONTSEmail( _
strFrom, _
strTo, _
strSubject, _
strBody, _
intFormat _
)
    Dim objNewMail
    ' létrehozunk egy CDONTS.NewMail objektumot
    Set objNewMail = _
    Server.CreateObject("CDONTS.NewMail")
    ' kitöltjük a tulajdonságokat
    objNewMail.From = strFrom ' feladó
    objNewMail.To = strTo ' címzett
    objNewMail.Subject = strSubject ' tárgy
    objNewMail.BodyFormat=intFormat ' formátum
    objNewMail.MailFormat=intFormat ' formátum
    objNewMail.Body = strBody ' az üzenet törzse
    On Error Resume Next
    ' elküldjük az üzenetet
    objNewMail.Send
    ' hibakezelés
    If Err.Number<>0 Then
        SendCDONTSEmail=False
    Else
        SendCDONTSEmail=True
    End If
    ' töröljük a CDONTS.NewMail objektumot
    Set objNewMail = Nothing
End Function
%>
```

Mint látjuk `SendCDONTSEmail` öt paraméterrel rendelkezik. Az első négy paraméter **String** típusú és a feladót, címzettet, tárgyat és az üzenet törzsét adják meg. A függvény utolsó paramétere, `intFormat`, megadja az elküldendő email formátumát: `0` HTML formátumnak, `1` sima szövegnek felel meg. A függvényt olyan egyszerűre szabtam, amennyire csak lehet. Szükség esetén könnyen bővíthető.

Azzal kezdjük, hogy a megszokott `Server.CreateObject` metódussal, egy `CDONTS.NewMail` objektumot hozunk létre. Ez az objektum testesíti meg az email-t. Minden email-re egy új `CDONTS.NewMail` objektumot kell használjunk. A komponens DLL-je a `winnt\system32` mappába található `cdonts.dll` állomány.

A `CDONTS.NewMail` objektum több tulajdonsággal rendelkezik. Itt csak a `From`, `To`, `Subject`, `BodyFormat`, `MailFormat` és `Body` tulajdonságokat használjuk. Ezeken kívül léteznek még a `Cc`, `Bcc` és más tulajdonságok, melyeket most nem használunk, de könnyen felvehetők a függvénybe. A `BodyFormat` és `MailFormat` tulajdonságokon kívül, a többi értéke karakterláncok.

A `BodyFormat` tulajdonság `0` vagy `1` értékeket veheti fel és meghatározza az üzenet törzsének (*body*) formátumát. A HTML formátumnak `0`, egy sima szövegnek `1` felel meg. A `MailFormat` tulajdonság is csak két értéket kaphat: `0` - a `CDONTS.NewMail` objektum formátuma egy MIME típus, 1 - a `CDONTS.NewMail` objektum formátuma sima szöveg. Ha HTML formátumú üzenetet akarunk küldeni, akkor a `BodyFormat` és `MailFormat` tulajdonságokat `0`-ra kell állítanunk. Sima szöveg esetén, elég ha `BodyFormat = 1`, mert `MailFormat` alapértelmezett értéke 1.

Miután az `objNewMail` `CDONTS.NewMail` típusú objektum szükséges tulajdonágaihoz hozzárendeltük a kellő értékeket, elküldhetjük a üzenetet az objektum `Send` metódusával. A `Send` metódus szintaxisa:

```
objNewMail.Send( _
[From][, To][, Subject][, Body][, Importance] _
)
```

s mint látjuk, mindegyik paramétere opcionális. A paraméterek az azonos nevű tulajdonságnak felelnek meg és az előbbiek, a `To` kivételével felülírják az utóbbiakat.

A `To` tulajdonság, illetve paraméter értékeként megadhatunk egy karakterláncot több címzettel, mindegyik címét ; (pontvessző) karakterrel elválasztva. A `To` paraméter listája nem írja felül a `To` tulajdonság listáját. A `Send` metódus mindkét lista tagjához elküldi az üzenetet, ezért ügyelnünk kell, nehogy egyes címzett kétszer szerepeljen az összesített listában.

Az `Importance` paraméteren és tulajdonságon kívül (létezik egy ilyen tulajdonság is), egyik tulajdonságnak és paraméternek sincs alapértelmezett értéke. A `Subject` és `Body` tulajdonságok, illetve paramétereken kívül, a tulajdonságok és paraméterek értéke nem lehet az üres karakterlánc, mert akkor a `Send` metódus egy hibát generál. Az `Importance` paraméter és tulajdonság alapértelmezett értéke `1` (normális fontosságú), további értéke `0` (alacsony fontosságú) vagy `2` (magas fontosságú) lehet.

A `CDONTS.NewMail` objektum más metódussal is rendelkezik. Itt csak az `AttachFile` metódust említem meg, de tárgyalása messzire vezetne, ezért mellőznünk kell. Akit a téma érdekel, az próbálja ki a fenti függvény működését a `CDONTS/CDONTSSendEmail_vbs.asp` állománnyal. Hogy az alkalmazás tényleg működjön egy SMTP szerverre van szükség. Aki nem rendelkezik egy működőképes SMTP szerverrel, az kipróbálhatja a példaalkalmazás *online* változatát, amit a webhelyemen érhet el (`www.hatvany-online.net/aspclassic`).

A bejelentkezés után a látogató kipróbálhatja a `4. fejezet/CDONTS` mappában található `CDONTSSendEmail_vbs.asp` példaalkalmazást, amivel saját magának egy email-t küldhet. A HTML formátumú email, az MSDN Library CDONTS komponensre vonatkozó lapokhoz vezet.

4.12 A CDOSYS email-komponens

Windows 2000 egy új email-komponenssel gazdagítja a rendelkezésünkre álló szerverkomponensek sorát. A CDOSYS komponensről van szó.

A CDOSYS komponens objektummodellje sokkal bonyolultabb, mint a CDONTS komponensé és természetesen a Windows 2000 komponense sokkal többet is tud. Itt csak azt említem meg, hogy CDOSYS több SMTP vagy Exchange szervert tud kiszolgálni, melyek természetesen nem kell a webszerveren fussanak.

Ismét egy függvénybe csomagoltam mindazt, amire szükségünk van, hogy a CDOSYS komponens segítségével egy email-t elküldhessünk:

```
<%
' elküld egy email-t a CDOSYS komponens segítségével
' paraméterek:
' strFrom - feladó
' strTo - címzett
' strSubject - tárgy
' strBody - üzenet törzse
' intFormat - üzenet formátuma:
' =0 - HTML, =1 - szöveg
Function SendCDOSYSEmail( _
  strFrom, _
  strTo, _
  strSubject, _
  strBody, _
  intFormat _
)
%>
```

```
<!-- #include file="cdosys_vbs.inc" ------------>
<%
 Dim objMessage, objConfig

 ' létrehozzuk a CDO.Message és
 ' CDO.Configuration objektumokat
 Set objMessage = CreateObject("CDO.Message")
 Set objConfig = CreateObject("CDO.Configuration")

 ' Beállítás:
 ' az üzenetet a mailszerverhez küldjük
 objConfig.Fields(cdoSendUsingMethod) = cdoSendUsingPort '2
 ' Email-szerver
 objConfig.Fields(cdoSMTPServer) ="mail.domain.com"
 ' Email-port: 25 (SMTP szerver esetében)
 objConfig.Fields(cdoSMTPServerPort) = 25
 ' azonosítás az Email-szerveren
 objConfig.Fields(cdoSMTPAuthenticate) = cdoBasic '1
 ' azonosítjuk magunkat
 ' azonosító
 objConfig.Fields(cdoSendUserName)="name@domain.com"
 ' jelszó
 objConfig.Fields(cdoSendPassword)="password"
 ' frissítjuk a Fields gyujteményt
 objConfig.Fields.Update

 ' az objConfig CDO.Configuration objektumot
 ' az objMessage CDO.Message objektumhoz kötjük,
 ' azaz konfiguráljuk az email-t
 Set objMessage.Configuration = objConfig

 objMessage.To = strTo ' címzett
 objMessage.From = strFrom ' feladó
 objMessage.Subject = strSubject ' tárgy
 'az üzenet törzse
 If intFormat = 0 Then
       ' az üzenet HTML törzse
      objMessage.HtmlBody = strBody
 Else
       ' az üzenet szöveges törzse
      objMessage.TextBody = strBody
 End If
 ' hibakezelés:
 On Error Resume Next
 ' elküldjük az üzenetet
 objMessage.Send
 If Err.Number = 0 Then
       SendCDOSYSEmail=True
 Else
       SendCDOSYSEmail=False
 End If
End Function
%>
```

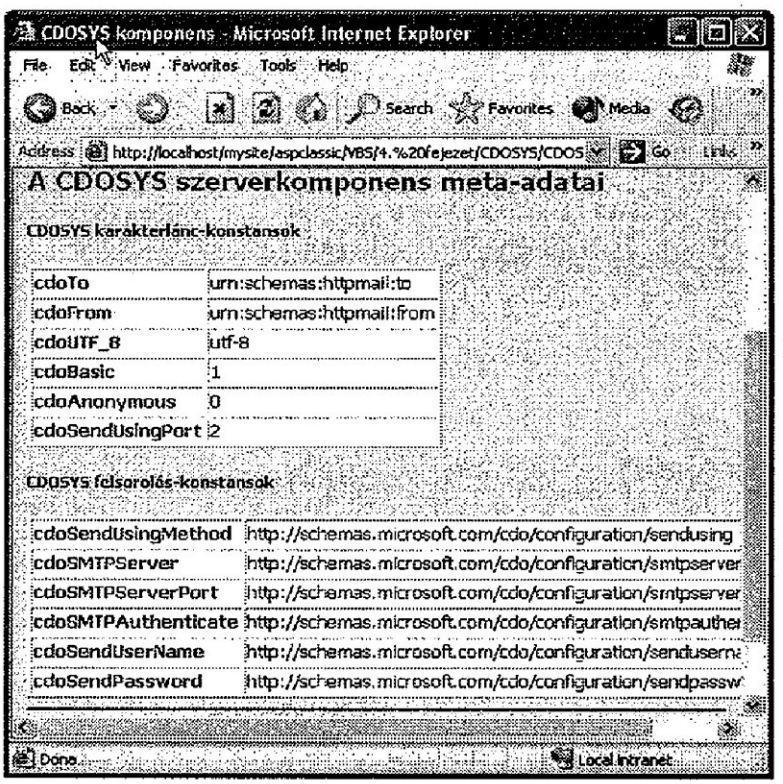
Mielőtt a függvényt megtárgyalnánk foglakozzunk a beillesztett `cdosys.inc` állománnyal!

A függvény több konstanst használ, mint például a `cdoSendUsingMethod` konstanst. A függvény forráskódjában könnyen felismerhetjük ezeket a konstansokat, mert azonosítójuk (*identifier*) `cdo` előtaggal rendelkezik. Ezek a konstansok a CDOSYS komponens típuskönyvtárában vannak definiálva. Természetesen beköthettük volna egy `<METADATA>` elemmel ezt a típuskönyvtárat is a függvényünk állományába. A dokumentáció szerint ezt a következő `<METADATA>` elemmel érhetjük el:

```
<!--METADATA TYPE="typelib"
UUID="CD000000-8B95-11D1-82DB-00C04FB1625D"
NAME="CDO for Windows 2000 Type Library" -->
```

amit nem könnyű megjegyezni. Sokkal könnyebb egy ASP lap segítségével a típuskönyvtárból a szükséges konstansokat kiolvasni. Az így kiolvasott konstansokat egy ASP lapban definiálhatjuk, amit szükség esetén beágyazhatunk állományunkba.

A `CDOSYS` mappába elhelyezett `CDOSYSMetadata_vbs.asp` lap kiolvassa ezeket a konstansokat. A lapot a 4.20. ábrán látjuk.



4.20. ábra A `CDOSYS` *komponens típuskönyvtárából kiolvasott konstansok*

Ezek után könnyen megírhatjuk a `cdosys_vbs.inc` lapot, amelyikben a fenti konstansokat definiáljuk.

Most nézzük meg közelebbről a függvényt! A CODSYS komponens esetében két objektumot kell létrehozzunk: a `CDO.Message` és a `CDO.Configuration` objektumokat. Az első az email szerkesztését és elküldését, a második az üzenet konfigurálását teszi lehetővé. A `CDO.Message` tulajdonságairól és a `Send` metódusáról nagyjában ugyanazt mondhatjuk el, mint a `CDONTS.NewMail` megfelelő elemeiről. Újak viszont a `HtmlBody` és `TextBody` tulajdonságuk, melyek szerepe viszont nevükből világosan kitűnik.

Mivelhogy a CDOSYS komponens több SMTP vagy Exchange szervert tud kiszolgálni, a `Message` objektumot konfigurálni kell: meg kell "mondani" neki melyik SMTP szerverre küldje az üzenetet, melyik portot használja és persze a feladónak azonosítania kell magát. Amint a függvényből kitűnik, mindezt a `Configuration` objektum `Fields` gyűjteménnyel és a megfelelő CDOSYS konstansokkal végezzük el. Miután a `Fields` gyűjteményt kellőképpen feltöltöttük, a `Fields.Update` metódus meghívásával az adatokat véglegesen az `objConfiguration Configuration` típusú objektumhoz kötjük, majd az így aktualizált `objConfiguration` objektumot az `objMessage` üzenetobjektumhoz kapcsoljuk:

```
' az objConfig CDO.Configuration objektumot
' az objMessage CDO.Message objektumhoz kötjük,
' azaz konfiguráljuk az email-t
Set objMessage.Configuration = objConfig
```

Ezután megadhatjuk az `objMessage` objektum még hiányzó tulajdonságait és elküldhetjük ez üzenetet.

A CDOSYS szerverkomponenssel befejeztük az úgynevezett Microsoft web- és email-komponenseinek tárgyalását. A következő fejezetben látni fogjuk, hogyan készíthetünk mi magunk ASP szerverkomponenst.

5. ASP komponensek előállítása

Ez előző fejezetekben megismertük az ASP belső objektumait, láttuk, hogyan kell az ASP `Scripting` objektumait használni és foglalkoztunk az MS web- ás email-komponensekkel. Elérkezett az idő, hogy megismerkedjünk az ASP komponensek előállításával.

Ebben a fejezetben először röviden áttekintjük a COM (*Component Object Model*) technológiát, majd az ASP komponensekkel foglalkozunk. Megismerkedünk a szkript és a Visual Basic komponensekkel és látni fogjuk, hogyan kell ezeket a komponenseket előállítani és alkalmazni. A fejezet végén az Olvasó rendelkezni fog mindazon ismerettel, amire az ASP komponensek előállításánál és alkalmazásánál szüksége lesz.

5.1 A COM (*Component Object Model*) technológia rövid áttekintése

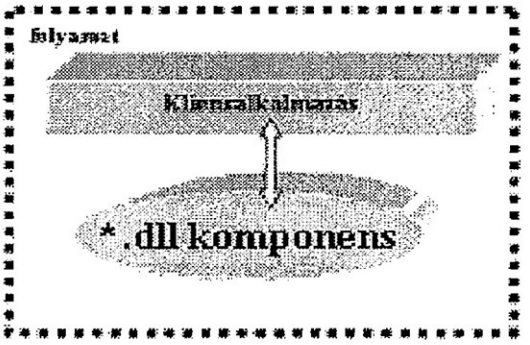
A mostanáig látottakból kiindulva, könnyen megállapíthatjuk, hogy a komponensek adta lehetőségek nagy részben leegyszerűsítik és meggyorsítják az ASP alkalmazások előállítását. Gondoljuk például arra, milyen könnyen lehet a `scrrun.dll` komponens segítségével, amely a `FileSystemObject` objektumot implementálja, állománymű-veleteket elvégezni. Ez a megállapítás nem csak az ASP alkalmazásokra, hanem bármilyen, nem feltétlenül a Windows operációs rendszer alatt futó alkalmazásra is érvényes. Egy technológia, mely lehetővé teszi bizonyos, gyakran megismétlődő feladatok megoldásának elkülönítését, külön állományokba való tárolását és ezen megoldások más alkalmazások által való ismételt felhasználását – egy nagyon jelentős szerepet tölthet be az alkalmazások programozásában.

Egy ilyen technológia a Windows alkalmazások részére a Microsoft által megalapozott COM vagy *Component Object Model* technológia. Microsoft 1993-ban vezette be a COM technológiát, mely egy modellspecifikáció a komponensek leírására, implementálására és használatára Windows platformokon.

A COM komponensek DLL vagy EXE bináris állományok, melyek egy bizonyos feladat megoldására eljárásokat, függvényeket és utasításokat tartalmaznak. A komponenseket e célra írt alkalmazások használhatják fel. Ebben a komponens – alkalmazás együttműködésben a komponens játssza a kiszolgáló (szerver) szerepét, míg a komponens szolgálatait igénybe vevő alkalmazás az ügyfél (kliens) szerepét tölti be. A COM alapja egy bináris specifikáció, mely leírja, miként kínálják fel (*expose*) a komponen-

sek szolgáltatásaikat, az általuk tárolt eljárásokat és függvényeket, és ugyanakkor előírja, miként használhatják az ügyfélalkalmazások a felkínált szolgáltatásokat.

A DLL komponensek és a komponenseket felhasználó kliensalkalmazások közti kapcsolatot erősen leegyszerűsítve az 5.1. ábra szemlélteti. A szerver szerepét betöltő DLL COM komponenst a kliensalkalmazás saját folyamatába (*process*) tölti. Itt történik az adatcsere a komponens és az alkalmazás között. Ezért nevezzük a DLL komponenseket *in-process* (folyamaton belüli) komponenseknek.



5.1. ábra A DLL komponensek és a kliensalkalmazás közti kapcsolat vázlata

Mivelhogy a COM egy bináris specifikáció, messzemenően nyelvsemleges. Bármelyik programozási nyelv segítségével, mely a COM specifikációt támogatja, létrehozhatók COM komponensek és fordítva, az ezen a nyelven írt alkalmazások felhasználhatják a COM komponenseket. A COM-kompatibilis nyelvek közé tartoznak többek között a Visual Basic, (Visual) C++, Visual J++, SmallTalk és Delphi.

Az eddig látott ASP komponensek mind COM komponensek és ebből kifolyólag, amennyiben nem használnak ASP specifikus elemeket, nem csak ASP alkalmazásokban, de más Windows alkalmazásokban is felhasználhatók. Ilyen például a `FileSystemObject` objektumot implementáló `scrrun.dll` komponens. Amint ezt az ASP alkalmazások esetében láttuk, a kliensalkalmazások a komponensek által implementált objektumot hozzák létre. Erre több lehetőség van. Az ASP alkalmazások esetében legtöbbször a `Server` objektum `CreateObject` metódusát használtuk:

```
Dim objObject
Set objObject = Server.CreateObject(library.class)
```

ahol *`library`* a könyvtár neve és *`class`* az osztály neve. A `scrrun.dll` komponens `FileSystemObject` objektum esetében ez a kódrészlet így alakul:

```
Dim objFSO
Set objFSO = _
Server.CreateObject("Scripting.FileSystemObject")
```

Mint látjuk, nincs semmilyen utalás a komponenst megtestesítő, esetünkben a `scrrun.dll`, állományra. Honnan „tudja" a kliensalkalmazás, esetünkben az ASP alkalmazás, melyik komponensről van szó? Honnan veszi elő a komponens által felkínált eljárásokat és függvényeket? Ezekre a kérdésekre keressük a feleletet a következőkben.

Mielőtt azonban továbbmegyünk, érdemes megjegyezni, hogy a COM technológia egy sokrétű és bonyolult technológia, melynek részletes leírása akár több kötetet is kitölthet. Ennek a könyvnek nem feladata a COM technológia kimerítő tárgyalása. Itt csak az ASP alkalmazások előállításánál feltétlenül szükséges ismeretekre szorítkozom.

A COM komponensek előállításukkor egy egyértelmű nevet kapnak, egy úgynevezett globálisan egyértelmű azonosítót (GUID, *globally unique identifier*). A komponensek esetében a globálisan egyértelmű azonosítót osztályazonosítónak (CLSID, *class identifier*) nevezzük. A CLSID, mint mindegyik GUID, egy 128 bitből álló kód, amit a komponens a forráskódja lefordításakor kap a fejlesztőrendszertől. A CLSID-t a Microsoft által készített eszközök állítják elő, melyek a COM kompatibilis nyelvek fejlesztői rendszereibe be vannak építve, de természetesen ezeken kívül is használhatók. Alapjuk a `CoCreateGuid` API rutin az `ole32.dll` rendszerkönyvtárból.

A CLSID részben esetleges (*random*) számokból, részben a fejlesztő géptől nyert információból (mint például a gép Ethernet kártya címe és a fejlesztés ideje 100 nanomásodpercnyi pontossággal) lesz nagyon bonyolult algoritmusok segítségével előállítva. Ez az eljárás garantálja az azonosító tényleges egyediségét.

Egy ilyen GUID azonosítóval a 4. fejezetben találkoztunk. Amikor a CDOSYS email-komponenssel foglalkoztunk, akkor láttuk, hogy a komponens által használt konstansokat egy típuskönyvtárból kell kiolvasnunk. A CDOSYS típuskönyvtárát akkor a következő `<METADATA>` elemmel adtuk meg

```
<!--METADATA TYPE="typelib"
UUID="CD000000-8B95-11D1-82DB-00C04FB1625D"
NAME="CDO for Windows 2000 Type Library" -->
```

melyben az UUID (*universal unique identifier*) értéke a típuskönyvtárt azonosító GUID.

A COM komponensek előállításakor kapott CLSID-t a fordító más információval együtt a komponens bináris állományába írja be. Miután egy COM komponenst a gépünkre telepítettünk, regisztrálnunk kell a komponenst. A regisztrálást a telepítő program automatikusan elvégezheti, de mint látni fogjuk, „kézzel" is elvégezhető. A regisztrálás alatt a komponens CLSID-ját a regisztráló program a komponens bináris állományából kiolvassa és beírja a Windows **Registry**-be. Az operációs rendszer így

számon tarthatja a regisztrált komponenseket és kérésre rendelkezésre bocsátja a kérést intéző alkalmazásnak.

Legtöbbször a komponensek a globálisan egyértelmű azonosítón kívül, tehát az osztályazonosítón kívül úgynevezett programazonosítóval (PROGID, *programmatic identifier*) is rendelkeznek. A PROGID-k egy könnyen megjegyezhető, bizonyos szabály szerint felépített szöveggel azonosítják a komponenseket. Természetesen a PROGID távolról sem biztosít egy egyértelmű azonosítást. Kétely esetben a rendszer kénytelen visszanyúlni a CLSID-re. A példaként említett `scrrun.dll` komponens esetében a PROGID `"Scripting.FileSystemObject"`. Nem valószínű, de nem is zárható ki, hogy valaki ugyanezzel a PROGID-vel egy komponenst hoz létre. Az általunk előállított komponensek programazonosítóját, bizonyos korlátokon belül mi magunk adhatjuk meg. Ha Visual Basic-ben állítunk elő egy komponenst, akkor a fejlesztőrendszer a programazonosítót ebben a formában állítja elő:

```
"ProjectName.ClassModulName "
```

ahol *ProjectName* a projekt neve és *ClassModulName* a komponenst definiáló osztályt tartalmazó modul neve. Mivelhogy a projektnév és a modulnév szabadon választható, előfordulhat, hogy két különböző komponens ugyanazt a programazonosítót kapja.

A COM technológia e rövid áttekintése után, térjünk most át az ASP komponensek előállításának tárgyalására. Több lehetőségünk van ASP komponenseket előállítani. Mi itt két módszerrel fogunk foglalkozni. Megvizsgáljuk, hogyan hozhatunk létre szkript komponenseket és hogyan állíthatunk elő Visual Basic komponenseket.

5.2 Szkript komponensek

Microsoft *Window Script Components* (WSC) elnevezéssel jelöli azt a technológiát, mely segítségével szkriptnyelvek felhasználásával könnyen létrehozhatók COM komponensek. Ezeket a komponenseket nevezzük szkript komponenseknek. Legújabban a *server scriptlet* elnevezést is használja Microsoft a szkript komponensek megnevezésére. WSC technológiában használható szkriptnyelv bármely olyan nyelv, mely támogatja a *Microsoft ActiveX Scripting* interfészt. Ezek közé tartozik JScript, Microsoft Visual Basic Scripting Edition (VBScript), PERLScript, PScript és Python.

A *Microsoft ActiveX Scripting* interfész alapja a *Windows Script Host* (WSH) skriptértelmező (*script interpreter*), melyet a Windows 2000 és Windows XP rendszer magába foglal. A WSH két szkriptnyelv értelmezőt tartalmaz, a VBScript és a JScript értelmezőt, de más szkriptnyelvek értelmezőgépével is kibővíthető.

Egy WSH szkript egy `.vbs` vagy `.js` kiterjesztésű állomány, mely egy VBScript, illetve JScript nyelven írt szkriptet tartalmaz. A WSH értelmező felismeri ezeket a kiterjesztéseket és a megfelelő szkriptértelmezővel értelmezi és végrehajtja a szkriptet. Ha például tudni szeretnénk, a WSH melyik verziója van a gépünkre telepítve, akkor egy állományba írjuk a következő egyszerű szkriptet:

```
MsgBox "Windows Script verzió: " & WScript.Version
```

és mentsük el például `WSHVersion.vbs` név alatt. Ha most kétszer rákattintunk az állományra, akkor a WSH az állomány kiterjesztéséről felismeri a VBScript szkriptet és ennek megfelelően értelmezi a benne talált utasításokat. Jelen esetben a WSH verzióját jeleníti meg a `MsgBox` párbeszédablakban.

Ebben a szakaszban a szkript komponensekkel foglalkozunk. Látni fogjuk, hogyan lehet szkript komponenseket előállítani, illetve ASP alkalmazásokban felhasználni. Mindezt egy egyszerű példával fogom elmagyarázni. Annak ellenére, hogy a példa egyszerű, mindent tartalmazni fog, amit a komponensektől általában elvárunk: egy olyan feladatra kínál megoldást, ami többször is, különböző alkalmazásban is előfordulhat. Példakomponensünket egyaránt használhatjuk egy Visual Basic alkalmazásban és egy ASP alkalmazásban is. Ez csak azért lehetséges, mert példánk nem használ sem ASP specifikus elemeket (mint például az ASP belső objektumai), sem Win32 specifikus elemeket (mint például a `MsgBox` vagy a `Save File` párbeszédablakok). Ez azonban nem mindig így van. A könnyebb megérthetőség kedvéért választottam egy „semleges" komponenst.

5.2.1 A feladat és megoldása WSH szkripttel

Mint említettem, az ASP komponensek előállításával kapcsolatos kérdéseket egy példakomponenssel fogom illusztrálni. A szóban forgó komponenst több módszerrel fogjuk előállítani és többféleképpen fogjuk felhasználni. Ezért jó előkészítőnek tartom, ha a komponens lényegét, azaz a feladatot, aminek megoldására a komponens készült, illetve a komponens által kínált megoldást, WSH szkript, pontosabban VBScript szkript alakban is megismerjük.

A komponenst egy nagyon egyszerű feladat megoldására fogjuk megírni. A feladata Celsius (C), Fahrenheit (F) és Kelvin (K) fokok közti átalakítás lesz.

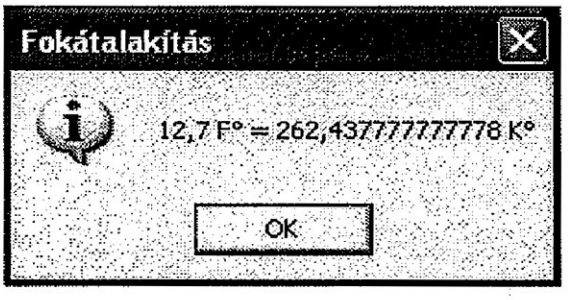
Oldjuk meg ezt a feladatot egyelőre egy VBScript szkript formájában, amit a `CFK1.vbs` állományba mentünk. A `CFK1.vbs` állomány a letöltött ZIP állományban a `CFK` mappában található. A `CFK1.vbs` szkript forráskódja a következő:

```
Option Explicit

Sub TransformDegree(strFrom, strTo, dblDegree)
  Dim dblResult

  Select Case UCase(strFrom)
    Case "C"
      Select Case UCase(strTo)
        Case "C"
          dblResult = dblDegree
        Case "F"
          dblResult = (9 / 5) * dblDegree + 32
        Case "K"
          dblResult = dblDegree + 273.16
      End Select
    Case "F"
      Select Case UCase(strTo)
        Case "C"
          dblResult = (5 / 9 ) * (dblDegree - 32)
        Case "F"
          dblResult = dblDegree
        Case "K"
          dblResult = (5 / 9) * (dblDegree - 32) + 273.16
      End Select
    Case "K"
      Select Case UCase(strTo)
        Case "C"
          dblResult = dblDegree - 273.16
        Case "F"
          dblResult =(9 / 5) * (dblDegree - 273.16) + 32
        Case "K"
          dblResult = dblDegree
      End Select
  End Select

  TransformDegree = dblResult

End Sub

Dim strFrom
Dim strTo
Dim dblDegree
Dim dblRes

strFrom = InputBox ("Celsius (C)," & vbCrLf & _
          "Fahrenheit (F) vagy" & vbCrLf & _
          "Kelvin (K)" & vbCrLf &  vbCrLf & _
          vbTab & vbTab "FOKRÓL?", "Fokátalakítás")
If Not (UCase(strFrom) = "C" _
        Or UCase(strFrom) = "F" _
        Or UCase(strFrom) = "K") Then
  MsgBox "Hibás adat!", vbCritical, "Fokátalakítás"
  strFrom = "C"
End If
```

```
strTo = InputBox("Celsius (C)," & vbCrLf & _
        "Fahrenheit (F) vagy" & vbCrLf & _
        "Kelvin (K)" & vbCrLf & vbCrLf & _
        vbTab & vbTab & "FOKRA?", "Fokátalakítás")
If Not (UCase(strTo) = "C" _
        Or UCase(strTo) = "F" _
        Or UCase(strTo) = "K") Then
  MsgBox "Hibás adat!", vbCritical ,"Fokátalakítás"
  strTo = "C"
End If

dblDegree= InputBox ("FOKSZÁM?", "Fokátalakítás")
If Not IsNumeric(dblDegree) Then
  MsgBox "Hibás adat!", vbCritical ,"Fokátalakítás"
  dblDegree = 0
End If

dblRes = TransformDegree (strFrom, strTo, dblDegree)

MsgBox dblDegree & " " & UCase(strFrom) & "°" & _
" = " & dblRes & " " & UCase(strTo) & "°" , _
vbInformation ,"Fokátalakítás"
```

Nézzük meg közelebbről a szkriptet. A szkript első részében a `TransformDegree` függvényt definiálja. A függvény három paramétert vár: `strFrom` – az átalakítandó fokrendszert azonosító betű, `strTo` – a cél-fokrendszert azonosító betű és `dblDegree` – a fokszám. A `TransformDegree` függvény a megadott paraméterektől függően elvégzi a szükséges átalakítást és visszaadja a kiszámított eredményt. A szkript második részében három egymás után meghívott `InputBox` segítségével összegyűjti a `TransformDegree` függvénynek szükséges paramétereket, majd meghívja a függvényt, mely elvégzi az átalakítást. A visszakapott eredményt egy `MsgBox` párbeszédablakban közli a felhasználóval.

Próbáljuk ki a szkriptet! Az eredményt az 5.2. ábrán láthatjuk.

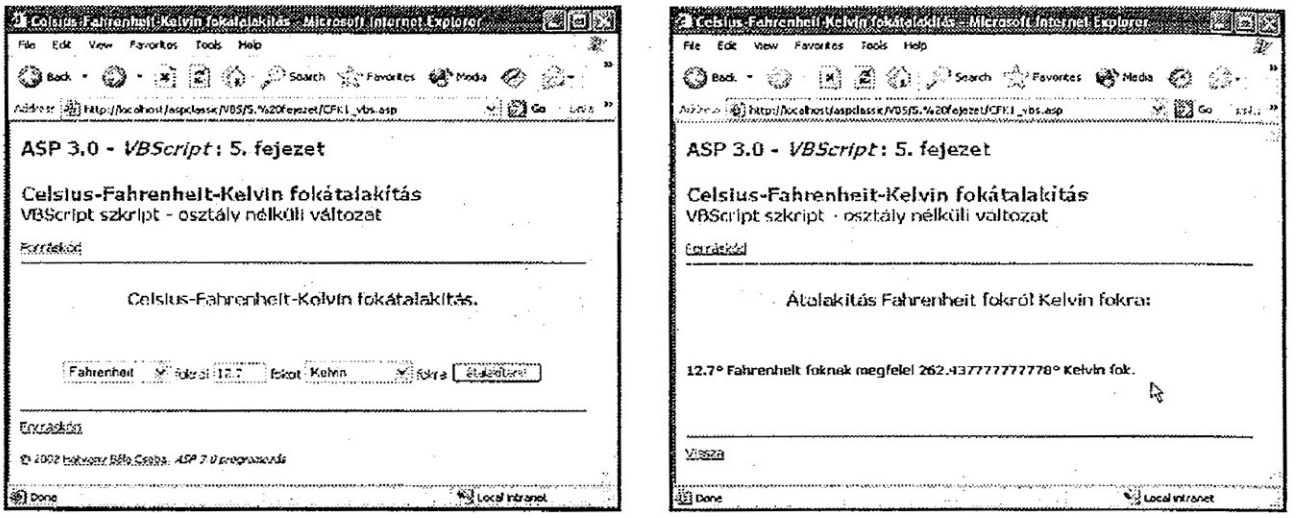


5.2. ábra Fokátalakítás VBScript szkripttel

A mostanáig tanultak alapján tudjuk, hogy a `TransformDegree` függvényt egy ASP állományban is definiálhattuk volna. A szkript második részét ezért egy kissé át kell

írnunk, mert ASP oldalakból nem hívhatjuk meg az `InputBox` és a `MsgBox` párbeszédablakokat. Hasonló párbeszédablakokat hívhatunk kliensoldali JavaScript szkripttel, de ha a HTML `<SELECT>` elemet használjuk, akkor legalább a fokrendszerek lekérdezésénél nem kell leellenőriznünk a látogató által beadott adat helyességét.

A letöltött ZIP állományban az 5. fejezetnek megfelelő mappa tartalmazza a `CFK-WSH1_vbs.asp` állományt, mely ezt a megoldást alkalmazza. Az eredményt az 5.3. ábrán láthatjuk.



5.3. ábra Az előbbi VBScript szkript alkalmazása egy ASP lapban

5.2.2 A feladat megoldása VBScript osztállyal

Az eddig látott példákban nem használtunk VBScript osztályokat. A hangsúlyt az ASP objektumok megismerésére és az ASP alkalmazások előállítására tettem, ezért a könnyebb megértés céljából, kerültem a bonyolultabb VBScript konstrukciókat, így az osztályokat is. Most viszont elérkeztünk egy ponthoz, ahol az osztályok használata elkerülhetetlen, ugyanis a komponenseket csakis osztályok felhasználásával állíthatjuk elő. Ezért ebben az alszakaszban röviden és szintén egy példa segítségével áttekintünk egy VBScript osztály szerkezetét.

Legjobb példa az imént megírt VBScript szkript, ezt fogjuk most egy osztály segítségével átfogalmazni.

Mint ismeretes egy osztály definícióját a következő utasítással adjuk meg:

```
Class ClassName
...
End Class
```

ahol *ClassName* az osztály neve vagy az osztályazonosító. Az osztályt definiáló változókat, tulajdonságokat és eljárásokat, egy szóval az osztálytagokat az utasítás kezdő és záró része közé kell írnunk.

Az osztályok példányait a memóriában objektumoknak nevezzük. A CName osztály egyik példányát, azaz egy CName típusú objektumot a **Set** és **New** utasításokkal hozzuk létre:

```
Dim objCName
Set objCName = New CName
```

A **Private** és **Public** kulcsszavakkal az osztálytagokat privát vagy nyilvános elérésűnek deklarálhatjuk. Minden osztálytag, melyet nem deklaráltunk privát vagy nyilvános elérésűnek, nyilvánosnak számit. A privát osztálytagok csak ezen az osztályon belül érhetők el (privát elérhetőség), a nyilvános osztálytagok elérhetők úgy az osztályon belül, mint az osztályon kívül is (nyilvános elérhetőség).

Az osztályok nyilvános változói az osztály tulajdonságainak foghatók fel. Az osztály tulajdonságait a **Property** utasítással is definiálhatunk. A

```
Property Let PropName ([args,] value)
...
End Property
```

utasítással egy tulajdonságot meghatározó eljárást adunk meg. A tulajdonság értékét a *value* paraméter adja meg. Az objCName objektum PName tulajdonágához a következő utasítással rendelhetünk egy értéket:

```
objCName.Pname = value
```

A

```
Property Get PropName [(args)]
...
   [Set] PropName = expression
End Property
```

utasítással kiolvashatjuk egy osztály tulajdonságát. A visszakapott értéket az *expression* kifejezés adja meg. Az objCName objektum PName tulajdonságát a következő utasítással kérdezhetjük le:

```
Dim var
var = objCName.Pname
```

E gyors áttekintés után nézzük meg, hogyan írhatjuk át az előbbi szkriptet osztályok segítségével.

Ha átvizsgáljuk a CFK1.vbs állományt, mindjárt világossá válik, hogy nincs szükségünk több osztályra, egy osztály definíciójával is átírhatjuk a szkriptet. Legyen ennek az osztálynak a neve CFKTransform.

A CFKTransform osztályon belül három privát változót definiálunk

```
Private m_fromDegree  ' eredeti fokrendszer
Private m_toDegree    ' cél-fokrendszer
Private m_degree      ' fokszám
```

Ezek a változók az eredeti fokrendszert, a cél-fokrendszert és a fokszámot fogják tartalmazni és velük fogjuk az átalakítást elvégezni.

Az eredeti fokrendszert a következő tulajdonsággal adjuk meg, illetve kérdezzük le:

```
' megadja az eredeti fokrendszert
Public Property Let FromDegree(newValue)
  If Not (UCase(newValue) = "C" Or _
          UCase(newValue) = "F" Or _
          UCase(newValue) = "K") Then
    MsgBox "Hibás adat!", vbCritical ,"Fokátalakítás"
    m_fromDegree = "C"
    Exit Property
  End If
  m_fromDegree = UCase(newValue)
End Property

' lekérdezi az eredeti fokrendszert
Public Property Get FromDegree()
  FromDegree = m_fromDegree
End Property
```

A cél-fokrendszert megadó, illetve lekérdező ToDegree tulajdonság forráskódja hasonló, míg a fokszámot meghatározó tulajdonságot a következő eljárásokba foglaltam:

```
' megadja a fokszámot
Public Property Let Degree(newValue)
  If Not IsNumeric(newValue) Then
    MsgBox "Hibás adat!", vbCritical ,"Fokátalakítás"
    m_degree = 0
    Exit Property
  End If
  m_degree = newValue
End Property

' lekérdezi a fokszámot
Public Property Get Degree()
  Degree = m_degree
End Property
```

E három privát változó és e három pár **Property** eljárással meghatározhatjuk a CFKTransform osztály tulajdonságait: az eredeti fokrendszert, a cél-fokrendszert és az átalakítandó fokszámot.

A fokszám átalakítására a következő függvényt használjuk:

```
Public Function TransformDegree()
  Select Case m_fromDegree
    Case "C"
      Select Case m_toDegree
        Case "C"
          TransformDegree = m_degree
        Case "F"
          TransformDegree=(9 / 5) * m_degree + 32
        Case "K"
          TransformDegree=deg + 273.16
      End Select
    Case "F"
      Select Case m_toDegree
        Case "C"
          TransformDegree=(5 / 9 ) * (m_degree - 32)
        Case "F"
          TransformDegree = m_degree
        Case "K"
          TransformDegree=(5 / 9) * (m_degree - 32) + 273.16
      End Select
    Case "K"
      Select Case m_toDegree
        Case "C"
          TransformDegree=m_degree - 273.16
        Case "F"
          TransformDegree=(9 / 5) * (m_degree - 273.16) + 32
        Case "K"
          TransformDegree = m_degree
      End Select
  End Select
End Function
```

Ezzel a CFKTransform osztály definíciója be is fejeződött. Ahhoz, hogy a CFKTransform osztállyal egy átalakítást végezzünk el, az osztálynak egy objektumát kell létre- hoznunk:

```
Dim objCFK
Set objCFK = New CFKTransform
```

Ezután már csak értékeket kell adnunk az objCFK objektum tulajdonságainak

```
objCFK.FromDegree = InputBox ("Celsius (C)," & _
vbCrLf & "Fahrenheit (F) vagy" & vbCrLf &  _
"Kelvin (K)" & vbCrLf &  vbCrLf & vbTab & vbTab &_
"FOKRÓL?", "Fokátalakítás")
```

```
objCFK.ToDegree = InputBox ("Celsius (C)," & _
vbCrLf & "Fahrenheit (F) vagy" & vbCrLf & _
"Kelvin (K)" & vbCrLf & vbCrLf & vbTab & vbTab &_
"FOKRA?", "Fokátalakítás")

objCFK.Degree= InputBox ("FOKSZÁM?", "Fokátalakítás")
```

és meg kell hívnunk az objektum `TransformDegree` metódusát

```
MsgBox objCFK.Degree & " " & _
       objCFK.FromDegree & "°" & " = "  & _
       objCFK.TransformDegree & " " & _
       objCFK.ToDegree & "°", _
       vbInformation, "Fokátalakítás"
```

A megoldás új alakját a `CFK` mappába elhelyezett `CFK2.vbs` állomány tartalmazza. Próbáljuk ki és győződjünk meg, hogy ugyanúgy alakítja át a fokokat, mint a `CFK1.vbs` szkript.

Természetesen, a kellő módosításokkal, ezt a szkriptet is egy ASP oldalból is végrehajthatjuk. A `MsgBox` és az `InputBox` párbeszédablakokat nem hívhatjuk meg egy ASP lapból. A bemeneti adatokat `<SELECT>` HTML elemekkel, illetve egy szövegmező értékéből nyerjük, az `objCFK` objektum tulajdonságainak továbbadjuk, meghívjuk az objektum `TransformDegree` metódusát és megjelenítjük a visszaadott eredményt.

Ezt az eljárást alkalmazza az `5. fejezet/CFK-WSH2_vbs.asp` állomány.

5.2.3 A Scriptlet Wizard

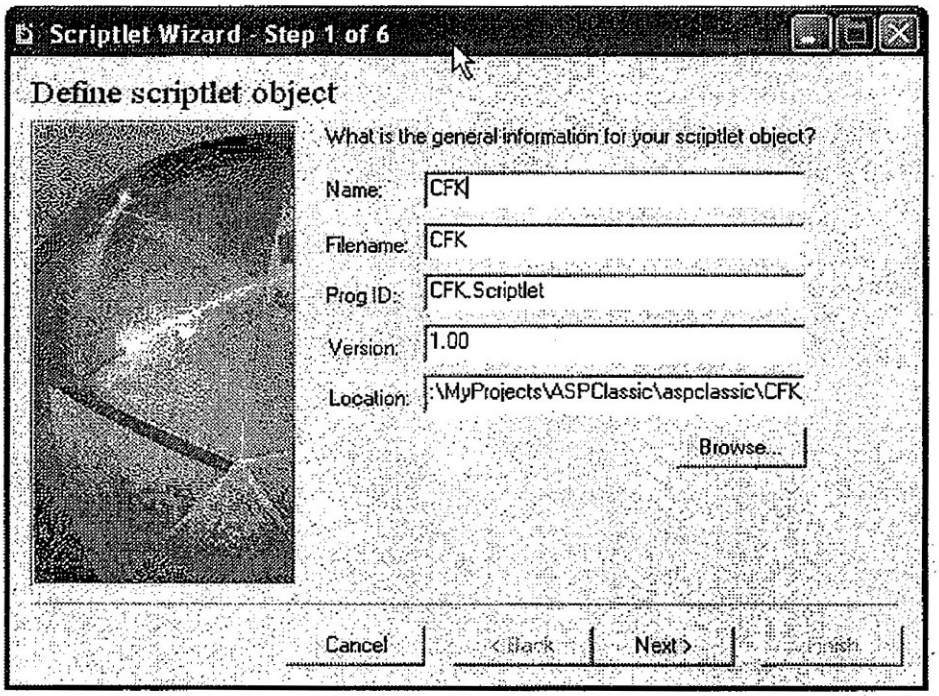
Miután láttuk, hogyan oldhatjuk meg a fokátalakítást VBScript osztályok segítségével, nézzük most meg, hogy hozhatunk létre egy szkript komponenst. Erre a célra Microsoft egy segítőeszközt bocsát a fejlesztők részére, a Scriptlet Wizard elnevezésű varázslót. A fejezet írásakor (2003 március) a Scriptlet Wizardnak egyelőre csak a beta 1 verziója készült el.

A segítőeszközt telepítő programot a Microsoft honlapjáról tölthetjük le (http://www.microsoft.com/downloads/details.aspx?FamilyId=408024ED-FAAD-4835-8E68-773CCC951A6B&displaylang=en), illetve megtaláljuk a letöltött ZIP állományban. Telepítsük a programot a gépünkre, majd a **Start/Programs/Microsoft Scriptlets/Microsoft Scriptlet Wizard** menüponttal indítsuk el.

A varázsló öt lépésben előállít egy `.sct` kiterjesztésű állományt, mely az általunk megadott adatok alapján egy szkript komponens vázlatát képezi. Az első párbeszéd-

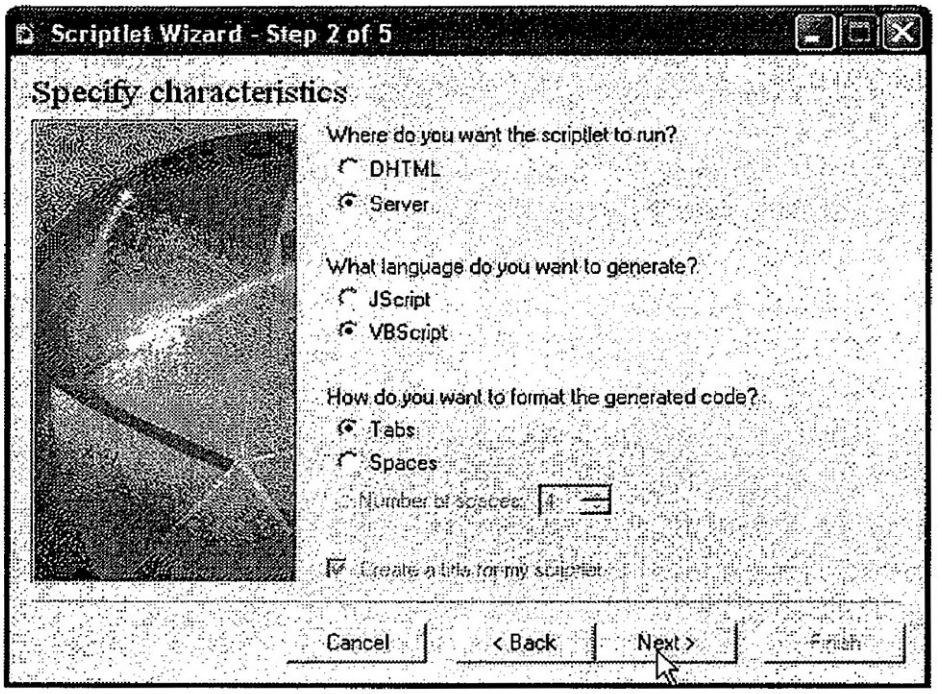
ablak (lásd az 5.4. ábrát) felszólít, hogy adjuk meg a komponens nevét, a komponenst tartalmazó állomány nevét, a programazonosítót, a komponens verzióját és a létrehozandó állomány helyét a merevlemezen. Legyen a komponensünk neve CFK. Mihelyt ezt beadtuk, a varázsló kitölti a többi adatot. Válasszuk ki az állomány helyét a **Browse...** parancsgombbal, majd nyomjuk meg a **Next >** gombot.

A második párbeszédablakon (lásd az 5.5. ábrát) a komponens jellemzőit adjuk meg. Itt válasszuk ki a **Server**, a **VBScript** és a **Tabs** választógombot. A **Server** választógombbal megadjuk, hogy egy szerverkomponenst akarunk létrehozni. A DHTML választógombbal egy klienskomponens előállítására állíthatjuk be a varázslót, ezzel az esettel azonban itt nem foglalkozunk. A **VBScript** választógombbal a szkriptnyelvet adjuk meg. A **Tabs** és a **Spaces** választógombokkal a szkripten belül a bekezdések behúzását adjuk meg. Ennek csupán formai jelentősége van. Nyomjuk meg a **Next >** gombot, hogy átkerüljünk a varázslóeszköz következő oldalára.

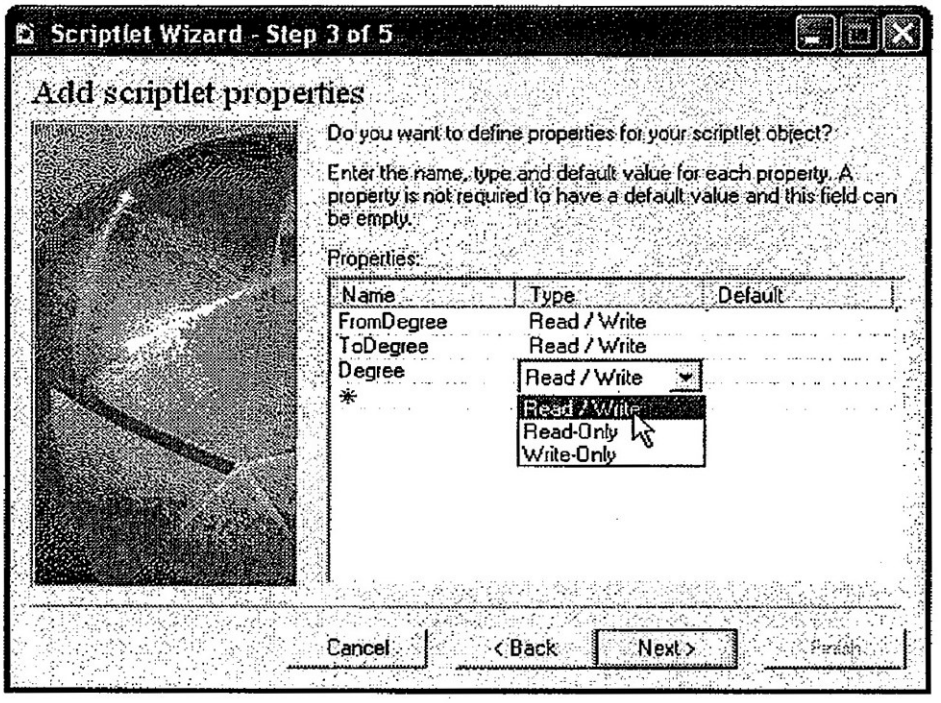


5.4. ábra Egy szkript komponens előállítása – első lépés

A harmadik ablak (lásd az 5.6. ábrát) a tulajdonságok definiálására szólít fel. Ettől a ponttól kezdve világosan látszik, hogy a varázsló egy osztály elemeit kéri tőlünk. Ez az ablak az osztály tulajdonságait kéri. Adjuk be a lista **Name** nevű oszlopába a három tulajdonság nevét: `FromDegree`, `ToDegree`, `Degree`. A második oszlop a tulajdonságok típusát rögzíti. Ha ebben az oszlopban egy cellára kattintunk, akkor egy legördülő listából kiválaszthatjuk a tulajdonság típusát. Mindhárom tulajdonság olvasható-írható, ezért válasszuk a **Read/Write** listaelemet. A harmadik oszlopban esetleges alapértelmezett értékeket adhatunk be. A mi esetünkben nincsenek ilyenek, ezért üresen hagyjuk ezt az oszlopot. Nyomjuk meg a **Next >** gombot, hogy a negyedik párbeszédablakhoz jussunk.



5.5. ábra Egy szkript komponens előállítása – második lépés

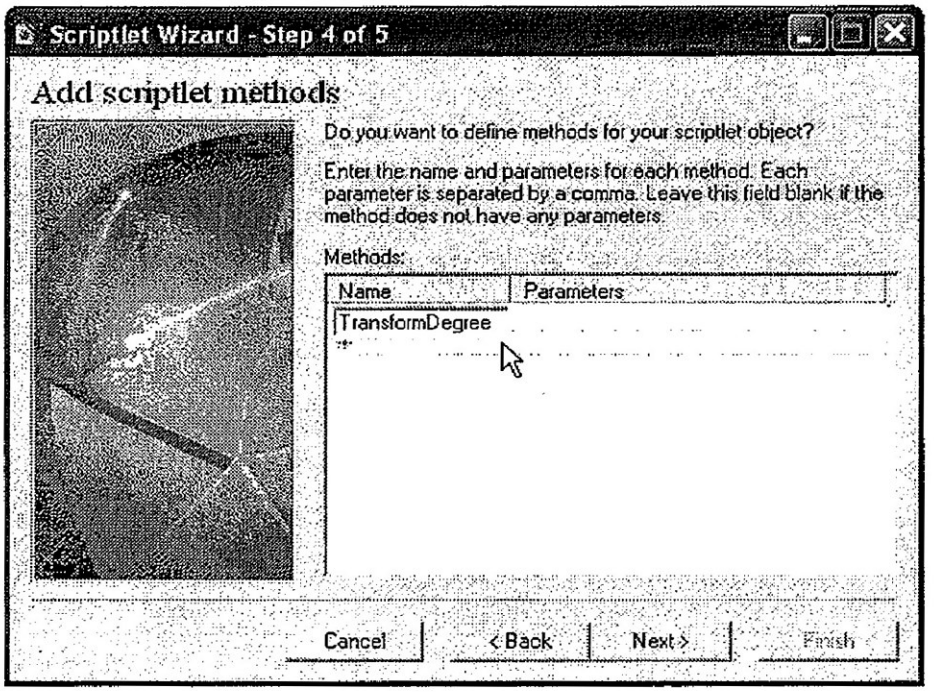


5.6. ábra Egy szkript komponens előállítása – harmadik lépés

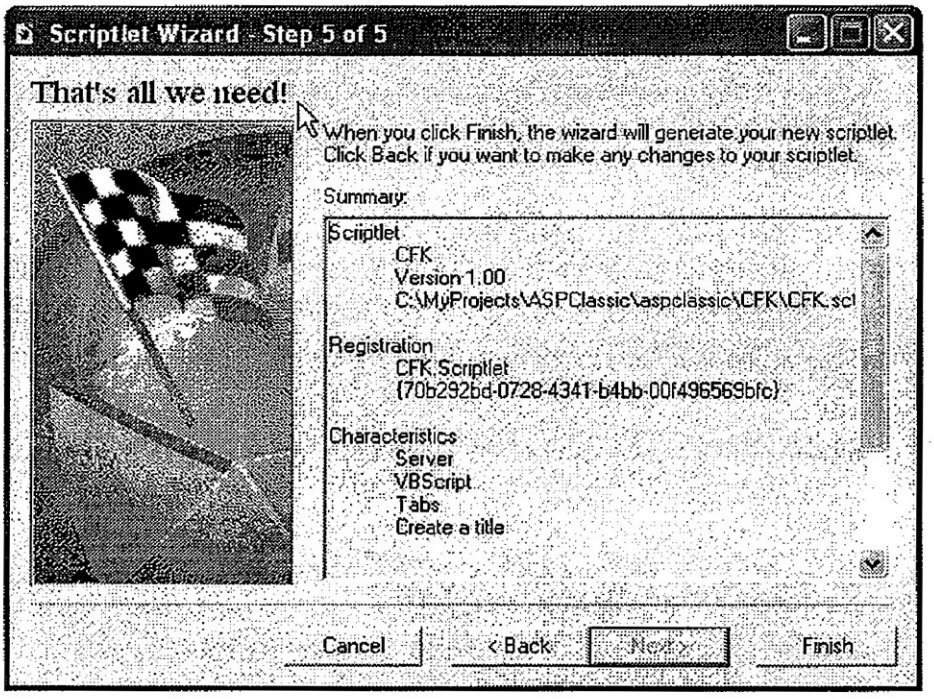
Ezúttal az osztály metódusait kell meghatároznunk (lásd az 5.7. ábrát). Osztályunknak csak egy metódusa van, a `TransformDegree` függvény, melynek nincs paramétere. Ezért adjuk be a metódus nevét, hagyjuk a **Parameters** oszlopot üresen és nyomjuk meg a **Next >** gombot.

Az ötödik ablak (lásd az 5.8. ábrát) összefoglalja a mostanáig beadott információt. Mindegyik ablakról a **< Back** gombbal visszamehetünk az előzőre, illetve a **Cancel**

gombbal megszakíthatjuk a program végrehajtását. Ha az utolsó ablakon megnyomjuk a **Finish** gombot, a program jelenti, hogy sikeresen létrehozta a `CFK.sct` állományt.



5.7. ábra Egy szkript komponens előállítása – negyedik lépés



5.8. ábra Egy szkript komponens előállítása – ötödik lépés

„Ez minden, amire szükségünk van!" – jelenti az utolsó párbeszédablak és nyilván igaza van: a szkript komponens vázát a program a `CFK.sct` állományba mentette. Most nézzük meg a Scriptlet Wizard által gyártott állományt.

5.2.4 A szkript komponens felépítése

A `CFG.sct` állományt bármilyen ASCII szövegszerkesztővel megnézhetjük. Nézzük meg a létrejött állományt!

Egy XML állományról van szó. XML vagy Extensible Markup Language (Kiterjesztett jelölő nyelv) egy a World Wide Web Consortium (W3C) által szabványosított jelölő nyelv (*markup language*), mely eredetileg az elektronikus dokumentumok megjelenítésére készült, ma azonban napról napra növekvő jelentőséget nyer a legkülönbözőbb területeken, ahol az adatcsere egy központi szerepet játszik. Az XML szabvány aránylag egy fiatal szabvány: 2003. február 10-én ünnepelte a W3C fennállásának ötödik évfordulóját.

Ennek a könyvnek nem célja az XML állományok tárgyalása. Az érdeklődő Olvasó a W3C honlapján (http://www.w3.org/XML/) részletes információt talál. Annyit azonban észre vehetünk, hogy az XML állományok felépítése hasonlít a HTML állományok felépítésére, amennyiben mindkettő jelölő nyelv, azaz jelölő elemekből állnak. Míg a HTML jelölő elemeit a nyelv szintaxisa rögzíti, addig az XML jelölő elemei szabadon definiálhatók. A HTML állománytól eltérően, az XML állományok megkövetelik, hogy minden jelölő elemnek egy záró része is legyen

Olyan XML állományok esetében, melyek definiált típusú információt közvetítenek, vagy másképpen mondva, definiált típusú dokumentumokat írnak le, a jelölő elemek rögzítettek, viszont nem a nyelv szintaxisa, de a hordozott információtípus, azaz a leírt dokumentumtípus szemantikája részéről.

Az `.sct` állományok jelölőelemeit is a szkript komponenst definiáló információ közvetítése céljából adta meg a Microsoft. A következőkben ezeket az elemeket vizsgáljuk meg, különös tekintettel az ASP szkript komponensekre.

Az állományt bevezető kódsor

```
<scriptlet>
```

arról ad hírt, hogy egy *scriptlet*-tel van dolgunk. A *scriptlet*-ek szerver komponenst vagy DHTML komponenst definiáló szkriptek. Esetünkben egy szerver komponensről van szó.

A következő elem az állományunkban a `<registration>` elem. A `<script>` elemtől eltérően, a `<registration>` elem üres vagy tartalomnélküli, azaz a nyitó és záró része között nem szerepel semmi. Ilyen és hasonló üres elemeket lerövidítve is írhatunk: `<registration />`. A `<registration>` elem viszont több attribútummal vagy jellemzővel rendelkezik:

```
<registration
      Description="CFK"
      ProgID="CFK.Scriptlet"
      Version="1.00"
      ClassID="{70b292bd-0728-4341-b4bb-00f496569bfc}"
>
</registration>
```

A `<registration>` elem a komponens regisztrálásához szükséges információt tárol az attribútumaiba.

A `Description` attribútum opcionális és a komponens leírását tartalmazza. A többi információval együtt ez a leírás is a **Regisrty**-be kerül, ahonnan más programok lekérdezhetik. A Visual Basic 6–ban például az **Object Browser** használja a `Description` attribútumba tárolt leírást. A `Description` attribútum értékét szabadon választhatjuk. Mi a Scriptlet Wizard első párbeszédablakán a **Name** szövegmezőbe adtuk be (lásd az 5.4. ábrát).

Ugyanitt adtuk be a következő két attribútum értékét is, a `ProgID` és a `Version` attribútumok értékét, melyek a komponens PROGID programazonosítóját és a verzióját határozzák meg.

Végül a `ClassID` attribútum a komponens CLSID osztályazonosítóját definiálja. Ezt természetesen nem lettünk volna képesek mi magunk megadni. A Scriptlet Wizard gyártott számunkra egy egyértelmű azonosítót. Gyakran szokott előfordulni, hogy a szkript komponensek fejlesztői csak a CLSID előállítására használják a Scriptlet Wizard programot, magát a `.sct` állományt pedig „házilag" készítik elő.

A `<registration>` elem eredetileg üres törzsébe, `<script language =` *`language_name`*`>` és `</script>` elemek közé egy szkriptet definiálhatunk.

A `<script>` elemmel lejjebb még foglalkozni fogunk. Egyelőre jegyezzük meg, hogy a *`language_name`* karakterlánc a használt szkriptnyelvet adja meg.

A `<registration>` elem törzsébe foglalt szkripten két függvényt definiálunk: a `register` függvényt és az `unregister` függvényt. A `register` függvény a regisztráció, az `unregister` függvény a regisztráció törlése alatt lesz végrehajtva. Töltsük ki a `<registration>` elem törzsét a következő függvényekkel:

```
<registration
      Description="CFK"
      ProgID="CFK.Scriptlet"
      Version="1.00"
      ClassID="{70b292bd-0728-4341-b4bb-00f496569bfc}
>
```

```
        <script language="VBScript">
          function register()
                MsgBox "Komponens regisztrálva!"
          end function

          function unregister()
                MsgBox "Regisztráció törölve!"
          end function
        </script>
    </registration>
    <implements id=Automation type=Automation>
```

Az <implements> elemen belül a komponens tulajdonságai és metódusai lesznek deklarálva. Az elem id = Automation és type = Automation attribútumaival a szkript megadja, hogy egy szerver komponens, nem pedig egy DHTML komponens definíciója következik. A mi komponensünknél ez a definíció az ismert FromDegree, ToDegree és Degree tulajdonságokból és a TransformDegree metódusból áll. A tulajdonságokat a <property> elemmel, a metódusokat a <method> elemmel definiáljuk. A definiált tulajdonság vagy metódus nevét a megfelelő elem name attribútummal adjuk meg. Mivelhogy mi a komponensünk tulajdonságait írhatónak és olvashatónak deklaráltuk (lásd az 5.6. ábrát), a tulajdonságokat deklaráló <property> elem a <get/> és <put/> elemeket tartalmazza. Egy csak olvasható tulajdonságot definiáló <property> elem nem tartalmazza a <put/> elemet, illetve egy csak írható tulajdonságot definiáló <property> elem nem tartalmazza a <get/> elemet.

A következő elem az .sct állományban egy <script> elem. Az <implements> elemben deklarált tulajdonságokat és metódusokat ebben a <script> elemben kell definiálnunk.

A szkript elején deklarált három változó a CFKTransform osztály három privát tagjának felel meg. Emlékeztetőül, ezt az osztályt az 5.2.2 alszakaszban definiáltuk. Az előbb látott tulajdonságok és a metódus is a CFKTransform osztály tulajdonságainak és metódusának felelnek meg. A tulajdonságok definíciójával nincs sok dolgunk. A Scriptlet Wizard előkészített számunkra mindegyik tulajdonság részére egy-egy vázlatot és ezek tökéletesen megfelelnek nekünk. A CFKTransform osztály tulajdonságai és következésképpen a készülő komponensünk tulajdonságai is egyszerűen beírnak, illetve kiolvasnak egy-egy értéket. Ha viszont olyan komponenst akarunk létrehozni, ahol a tulajdonságok a beadott értéket valamiféleképpen feldolgozzák, mielőtt beírják, akkor a Scriptlet Wizard által létrehozott tulajdonságvázlatot ki kell egészítenünk. Ugyanez a helyzet a metódusokkal is. Esetünkben a TransformDegree vázlata túl kevés a komponensünk részére, tehát ki kell egészítenünk.

Nézzük meg a CFKTransform osztály azonos nevű metódusát. Komponensünk TransformDegree metódusa ugyanazokat a számításokat végzi el. Ezért bővítsük a TransformDegree metódust a következőkkel

```
function TransformDegree()
  Select Case FromDegree
    Case "Celsius"
      Select Case ToDegree
        Case "Celsius"
          TransformDegree = Degree
        Case "Fahrenheit"
          TransformDegree=(9/5) * Degree + 32
        Case "Kelvin"
          TransformDegree= Degree + 273.16
       End Select
    Case "Fahrenheit"
      Select Case ToDegree
        Case "Celsius"
          TransformDegree=(5/9 ) * (Degree - 32)
        Case "Fahrenheit"
          TransformDegree = Degree
        Case "Kelvin"
          TransformDegree=(5/9) * (Degree - 32)+273.16
      End Select
    Case "Kelvin"
      Select Case ToDegree
        Case "Celsius"
          TransformDegree= Degree - 273.16
        Case "Fahrenheit"
          TransformDegree=(9/5) * (Degree - 273.16)+32
        Case "Kelvin"
          TransformDegree = deg
      End Select
  End Select
end function
```

Ezzel be is fejeztük a komponenst leíró CFL.sct állomány elemzését és szerkesztését. Véglegesítsük az elvégzett módosításokat.

A következőn alszakaszban megvizsgáljuk, miként működik a szkript komponens.

5.2.5 A szkript komponens működése

A szkript komponensek egyik jellemzője, hogy nem bináris formában, azaz DLL vagy EXE állományokban tárolják az alkalmazásoknak felkínált funkcionalitásukat, hanem egy ASCII formátumú szkript állományban. Hogy vesz tudomást a szkript komponens által felkínált tulajdonságokról és metódusokról egy kliensalkalmazás? Hogy találja

meg egy kliensalkalmazás a szkript komponenst, amit kiszolgálóként akar felhasználni? Ezekre a kérdésekre keressük a feleletet ebben az alszakaszban.

Láttuk, hogy a DLL komponenseknél a kliensalkalmazás saját folyamatába tölti a dinamikusan kötött könyvtárat és a folyamaton belül történik a kliens és a szerver szerepét betöltő komponens közötti kommunikáció. Hogy jön létre a kommunikáció az ASCII formátumú szkript komponens és a kliensalkalmazás között?

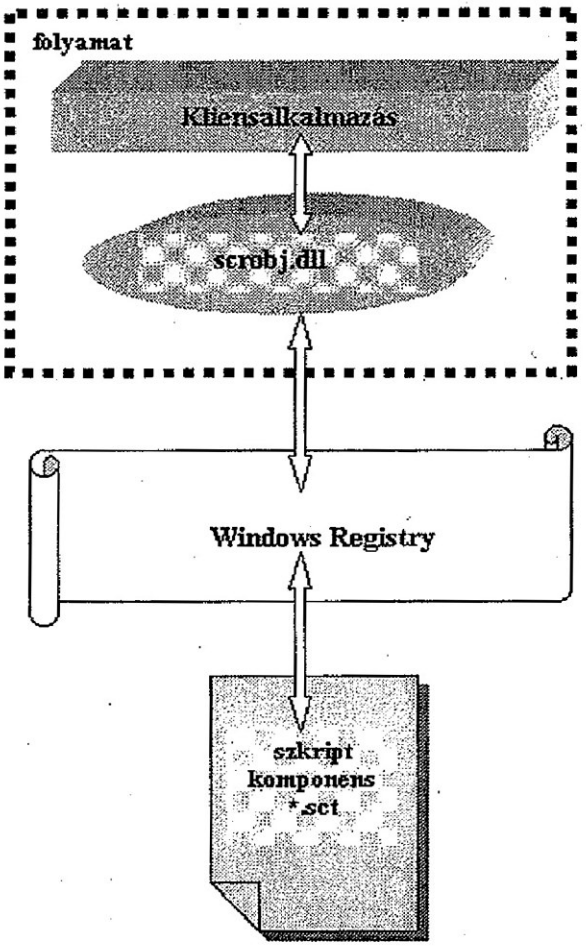
A Scriptlet Wizard telepítésével a gépünkre a programon kívül sok minden más is rákerült. Nézzük meg a mappa tartalmát, ahova a Scriptlet Wizard programot telepítettük! Itt többek között 35 HTML állományt és a `scrobj.dll` DLL-t fogunk látni. A HTML állományok egy rövid angol nyelvű bevezetőt tartalmaznak a *scriptlet*-ekről. A bevezető a `scplt.htm` állománnyal kezdődik, ahonnan a többi HTML lapra navigálhatunk. Minket azonban most a `scrobj.dll` állomány érdekel.

A szkript komponensek esetében a `scrobj.dll` állomány veszi át a DLL *in process* szerver szerepét, ezt az állományt tölti a folyamatába a kliensalkalmazás. A `scrobj.dll` komponens természetesen nem „tud" semmit, a mi komponensünkről, viszont „tudja" hogyan kell közvetítenie a kommunikációt a kliensalkalmazás és a szkript komponens között. Leegyszerűsítve a dolgokat, ezt úgy foghatjuk fel, hogy a `scrobj.dll` komponens elolvassa a szkript komponens szigorú szabályok szerint felépített tartalmát és kellőképpen feldolgozva, felajánlja a kliensalkalmazásnak. A `scrobj.dll` komponens így hozzáférést biztosít a kliensalkalmazásnak a szkript komponens összes tulajdonságaihoz és metódusaihoz.

Ezek után, már csak azt a kérdést kell tisztáznunk, hogy honnan szerez a `scrobj.dll` komponens tudomást a szkript komponens hollétéről? Hogy melyik komponensről van szó, azt a `scrobj.dll` a kliensalkalmazástól tudja meg. Hogy hol található ez a komponens, azt a `scrobj.dll` a Windows **Registry**-től tudja meg.

A kliensalkalmazás, a `scrobj.dll` és a szkript komponens közti kapcsolatot az 5.9. ábra mutatja be vázlatosan.

Azért, hogy mindez lehetséges legyen, szkript komponensünket regisztrálnunk kell a Windows **Registry**-be. A Scriptlet Wizard telepítésével a Windows **Registry**-be is változások álltak be. Az egyik ilyen változás a `.sct` kiterjesztésű állományokat illeti. Ha egy ilyen állományra, például az előbb létrehozott `CFK.sct` állományra rákattintunk az egér jobboldali gombjával, akkor a környezetmenüben három új menüpontot találunk: **Register**, **Unregister** és **Generate Type Library**. Minket most az első két menüpont érdekel, amivel regisztrálni lehet a szkript komponensünket, illetve a regisztrációt törölni lehet.



5.9. ábra A Kliensalkalmazás és a szkript komponens közti kommunikáció vázlata

Természetesen regisztrálhatjuk, illetve törölhetjük a regisztrációt a `regsvr32` utasítás beadásával a DOS konzolban is. Regisztrálásra adjuk be

```
regsvr32.exe scriptlet_path
```

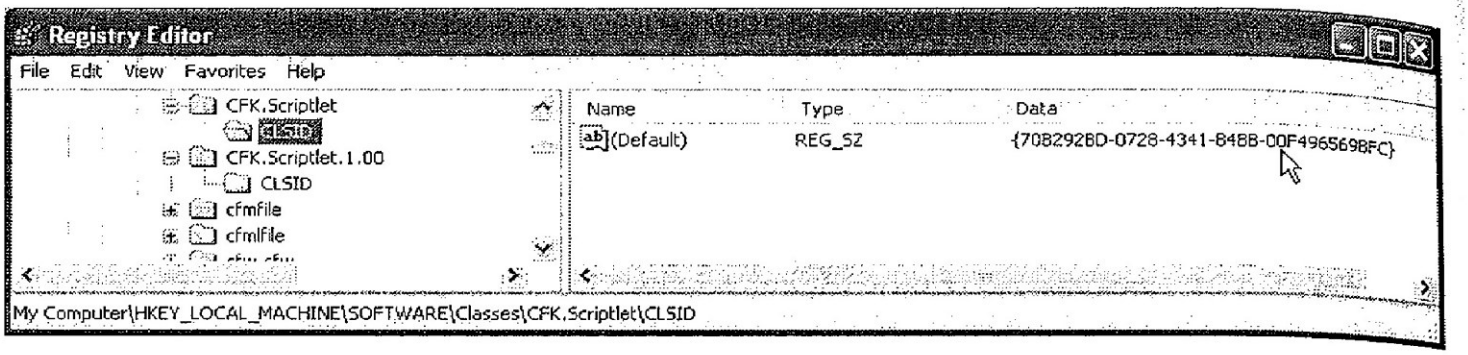
a regisztrálás törlésére pedig

```
regsvr32.exe /unreg scriptlet_path
```

ahol *`scriptlet_path`* a szkript komponens elérési útja.

Regisztráljuk a `CFK.sct` szkript komponenst. Ha a szkript komponenst hiba nélkül szerkesztettük meg, akkor a regisztrálás sikerrel jár. Nézzük meg mi került a **Windows Registry**-be.

Induljuk el a **Start/Run** menüből és a megjelenő ablakon adjuk be a `regedit` utasítást, amivel a **Registry** szerkesztőjét elindítjuk. Lapozzunk a `HKEY_LOCAL_MACHINE\SOFTWARE\Classes` mappához és keressük meg a `CFK.Scriptlet` mappát.



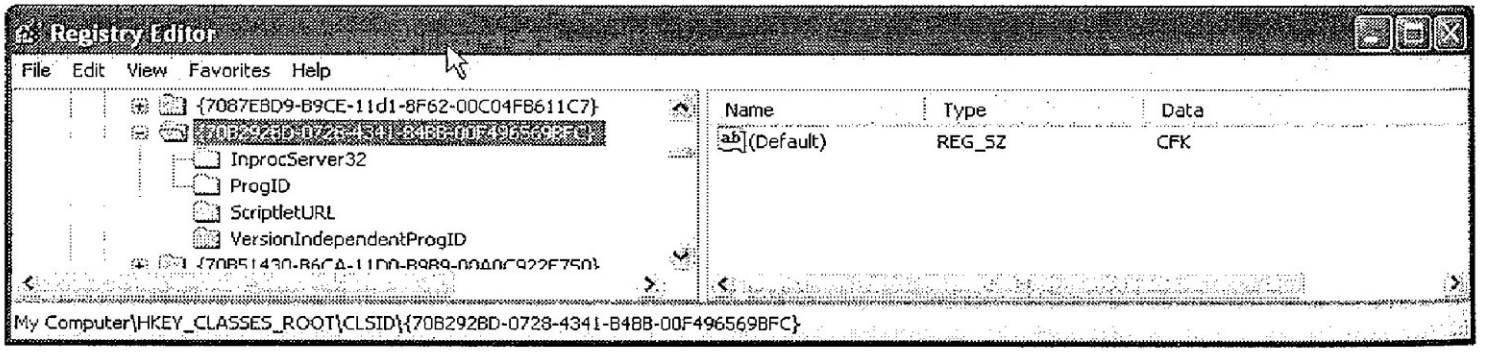
5.10. ábra A `CFK.Scriptlet` *szerver komponens a* **Regystry**-*ben*

Ennek a mappának van egy CLSID nevű almappája, melynek a tartalmát a szerkesztő jobb oldalán láthatjuk (lásd az 5.10. ábrát). A CLSID értéke megegyezik a Scriptlet Wizard által adott egyértelmű osztályazonosítóval.

Ugyanitt láthatjuk a `CFK.Scriptlet.1.00` mappát, mely tartalmában megegyezik a `CFK.Scriptlet` mappával, mivelhogy esetünkben csak egy verziót készítettünk a szkript komponensből. Ha több verziót készítünk a komponensből, akkor minden verzió egy külön mappába lesz regisztrálva és így könnyen hozzáférhetők lesznek.

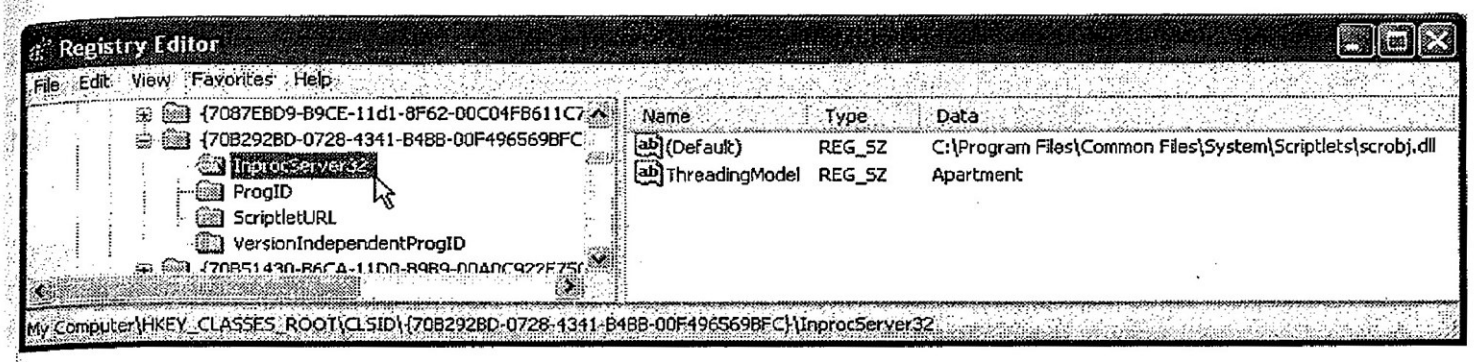
Egyelőre csak a komponensünk CLSID-jét láttuk a **Registry**-ben, de tudjuk, hogy a komponens elérési útja is itt van regisztrálva, különben a `scrobj.dll` nem tudja az információkat kiolvasni a komponensből. Ez az információ egy másik **Registry**-mappába van beírva. Lapozzunk a `HKEY_CLASSES_ROOT\CLSID` mappához és keressük meg a mi osztályazonosítónknak megfelelő almappát. A keresést megkönnyítjük, ha számon tartjuk, hogy az osztályazonosítók hexadecimális számok és az almappák növekvő sorrendben vannak felsorolva.

A `{70b292bd-0728-4341-b4bb-00f496569bfc}` osztályazonosítónak a mappáját az 5.11. ábrán láthatjuk. A mappának a `CFK` érték felel meg.



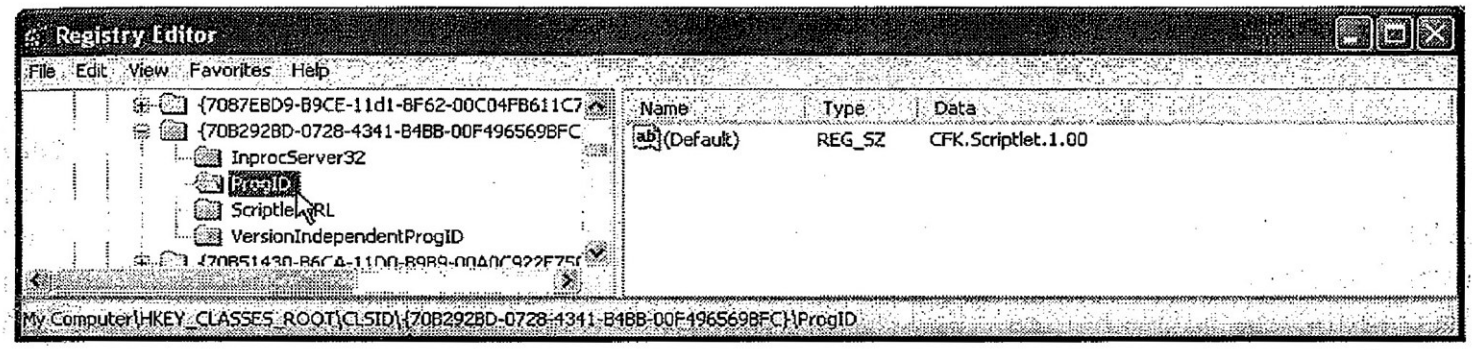
5.11. ábra A `CFK.Scriptlet` *szkript komponens bejegyzése a* **Registry**-*be*

A mappának négy almappája van: `InProcServer32`, `ProgID`, `ScriptletURL` és `VersionIndependentProgID`. Az `InProcServer32` az *in process* `scrobj.dll` szerver komponens elérési útját tartalmazza. Itt azt az elérési utat találjuk, ahová a Scriptel Wizard programot telepítettük (lásd az 5.12. ábrát).



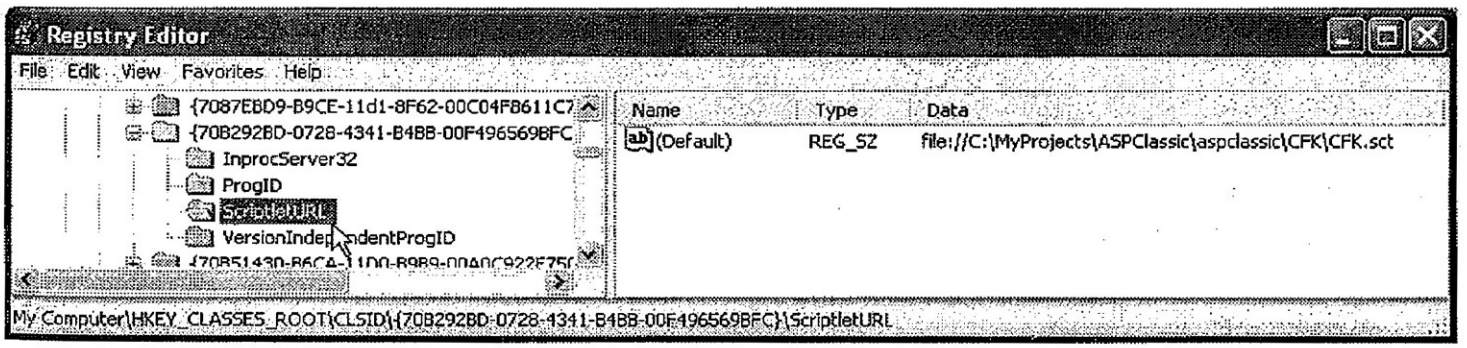
5.12. ábra A `scrobj.dll` *szerver komponens elérési útja*

A `ProgID` mappa (lásd az 5.13. ábrát) a komponensünk PROGID-jét tartalmazza, a verzió megadásával. Ezt a nevet kell használnunk a kliensalkalmazásokban, ha a komponensünknek megfelelő objektumot akarunk létrehozni és feltétlenül egy bizonyos verziót akarunk használni.



5.13. ábra A szkript komponens programazonosítója a verzió megadásával

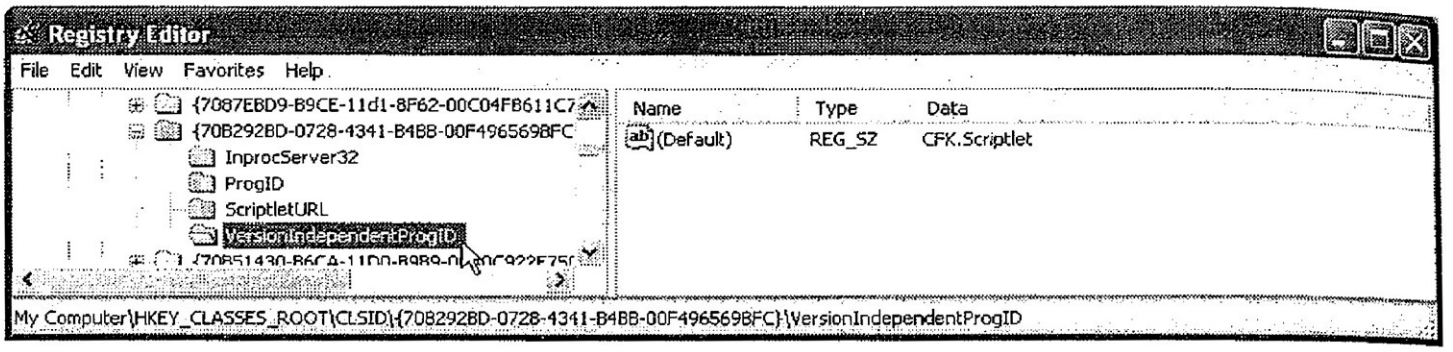
A `ScriptletURL` mappa a szkript komponens elérési útját tartalmazza (lásd az 5.14. ábrát).



5.14. ábra A szkript komponens elérési útja

Végül, a `VersionIndependentProgID` mappa (lásd az 5.15. ábrát) a komponensünk verziófüggetlen PROGID-jét tartalmazza. Ezt a nevet kell használnunk a kliens-alkalmazásokban, ha a komponensünk legújabb verziójának megfelelő objektumot akarunk létrehozni.

Mit látjuk, az 5.9. ábrán vázolt kapcsolatra szükséges összes információ tárolva van a **Registry**-ben, ami azt jelenti, hogy a komponensünket minden további nélkül kliens-alkalmazásokban fel tudjuk használni. Ezt fogjuk megvizsgálni a következő két alszakaszban.



5.15. ábra A szkript komponens verziófüggetlen programazonosítója.

5.2.6 A szkript komponens ellenőrzése Visual Basic programmal

A létrehozott szkript komponenst bármelyik COM komponens használatát támogató programban felhasználhatjuk. Ez azért lehetséges, mert a komponens nem használ ASP specifikus vagy a Win32 alkalmazásra specifikus eljárásokat. Mint említettem, ez természetesen nem mindig így van. ASP alkalmazásoknak szánt komponensek igenis használhatnak ASP specifikus eljárásokat. A gyakorlatban ez a legtöbbször előforduló eset. Esetünkben készakarva olyan feladat megoldására építettük fel a komponenst, hogy egyaránt használhassuk egy Win32 és egy ASP alkalmazásban.

Ebben és a következő alszakaszban ezt fogom megmutatni egy Visual Basic, illetve egy ASP példaalkalmazással. Ha a példák lényegét és működésüket megértettük, akkor könnyen szerkeszthetünk bonyolultabb feladatok megoldására alkalmas szerver komponenseket.

Lássuk most a Visual Basic példaalkalmazást! A projektünk egy ablakból áll, melyen két kombinált listát (*ComboBox*), egy szövegmezőt (*TextBox*) és egy parancsgombot (*Button*) helyeztem el. Az alkalmazás főablakát az 5.16. ábrán láthatjuk.

5.16. ábra A szkript komponenst ellenőrző Visual Basic alkalmazás főablaka

A parancsgomb eseménykezelő eljárásában hozzuk létre a szkript komponensnek megfelelő objektumot, lássuk el tulajdonságait a megfelelő értékekkel és hívjuk meg a fokátalakítást elvégző metódust:

```
Private Sub cmdTransform_Click()
  Dim objCFK As Object
  Dim FromDegree As String
  Dim ToDegree As String
  Dim dblResult As Double

  If cmdTransform.Caption = "átalakítani!" Then
    If IsNumeric(txtDeg.Text) Then
        ' létrehozzuk az objCFK objektumot
        Set objCFK = _
            CreateObject("CFK.Scriptlet")
        ' a objCFK tulajdonságaihoz
        ' hozzárendeljük a megadott értékeket
        objCFK.FromDegree = _
            cbFromDegree.Text
        objCFK.ToDegree = cbToDegree.Text
        objCFK.Degree = txtDeg.Text
        ' meghívjuk a objCFK objektum
        ' TransformDegree metódusát,
        ' hogy elvégezzük a kért átalakítást
        dblResult = _
            objCFK.TransformDegree()
        ' kiírjuk az átalakítás eredményét
        cbFromDegree.Visible = _
            Not cbFromDegree.Visible
        Label1.Visible = Not Label1.Visible
        txtDeg.Visible = Not txtDeg.Visible
        lblDeg.Visible = Not lblDeg.Visible
        cbToDegree.Visible = Not cbToDegree.Visible
        Label2.Visible = Not Label2.Visible
        lblResult.Visible = Not lblResult.Visible
        cmdTransform.Caption = "Vissza"

        With objCFK
            Caption = "Átalakítás " & .FromDegree & _
            " fokról " & .ToDegree & " fokra:"
```

```
            lblTitle.Caption = "Átalakítás " & _
            .FromDegree & " fokról " & .ToDegree & _
            " fokra:"
            lblResult.Caption = txtDeg.Text & _
            "° " & .FromDegree & _
            " foknak megfelel " & dblResult & _
            "° " & .ToDegree & " fok."
        End With
         ' felszabadítjuk a memóriát
        Set objCFK = Nothing
    Else
         ' txtDeg.Text nem egy elfogadott fokszám
        MsgBox "HIBÁS FOKSZÁM!" & vbCrLf & _
        "Kérem adjon be egy helyes fokszámot", _
        vbCritical, "CFK fokátalakítás"
    End If

  ElseIf cmdTransform.Caption = "Vissza" Then
    cbFromDegree.Visible = Not cbFromDegree.Visible
    Label1.Visible = Not Label1.Visible
    txtDeg.Visible = Not txtDeg.Visible
    lblDeg.Visible = Not lblDeg.Visible
    cbToDegree.Visible = Not cbToDegree.Visible
    Label2.Visible = Not Label2.Visible
    lblResult.Visible = Not lblResult.Visible
    cmdTransform.Caption = "átalakítani!"
    Caption = _
          "Celsius-Fahrenheit-Kelvin fokátalakítás."
    lblTitle.Caption = _
          "Celsius-Fahrenheit-Kelvin fokátalakítás."
  End If

End Sub
```

A szkript komponensnek megfelelő objektumot a

```
Set objCFK = CreateObject("CFK.Scriptlet")
```

utasítással hozzuk létre. Az `objCFK` objektumváltozót az eljárás elején deklaráltuk:

```
Dim objCFK As Object
```

A `CreateObject` eljárás `"CFK.Scriptlet"` paramétere megadja a szkript komponenst, amelynek egy objektumát akarjuk létrehozni. Mint látjuk, ez a paraméter hasonlóan van felépítve, mint a DLL komponenseket meghatározó paraméter, amit az 5.1 szakaszban láttunk. A könyvtár szerepét a szkript komponens neve tölti be, esetünkben `CFK`, az osztály neve pedig a szkript komponenseknél mindig `Scriptlet`.

Az objektum létrehozása után, a következő három utasítással értékekkel látjuk el az `objCFK` objektum `FromDegree`, `ToDegree` és `Degree` tulajdonságát, majd meghív-

juk az objektum `TransformDegree` metódusát

```
dblResult = objCFK.TransformDegree()
```

A forráskód hátramaradt része az eredmény megjelenítéséről gondoskodik. Miután az átalakítást a `dblResult` változóba mentettük, elrejtjük a `ComboBox` és `TextBox` vezérlőket és az eredményt kiírjuk egy címke (*label*) segítségével. Ugyanakkor megváltoztatjuk a parancsgomb feliratát és lehetővé tesszük, hogy a felhasználó egy újabb fokátalakításra adjon be adatokat. Az eredményt kiíró ablakot az 5.17. ábrán láthatjuk.

5.17. ábra A szkript komponenst ellenőrző Visual Basic alkalmazás fokátalakítása

A Visual Basic projekt a letöltött ZIP állományban a `CFK\CFKVBSCT` mappában található.

Miután meggyőződtünk, hogy a szkript komponens az elvárásoknak megfelelően működik, beépíthetjük egy ASP alkalmazásba.

5.2.7 A szkript komponens ASP alkalmazása

A szkript komponenst felhasználó ASP alkalmazás megírása ezek után már nem egy nehéz feladat. Az ASP lap szkript részében létre kell hoznunk egy objektumot, mely a szkript komponens megtestesítője a memóriába. Tudjuk, hogy erre a célra a `Server` objektum `CreateObject` metódusát kell használnunk.

Induljunk el a már látott `CFK-WSH2_vbs.asp` lapból. Ezzel az ASP lappal az 5.2.2. alszakaszban találkoztunk, amikor a megoldandó feladatot egy VBScript osztályba foglaltuk. Most tudjuk, hogy a szkript komponensünk is egy osztályba foglalja a feladat megoldását, tehát ha erről a lapról indulunk el, akkor valószínűleg keveset kell a forráskódon módosítani.

Másoljuk le a `CFK-WSH2_vbs.asp` lapot a `CFK-SCT_vbs.asp` név alatt! A `CFKTransform` osztály definíciójára most nincs szükségünk, mert mindaz, amit az osztályban definiáltunk, most a szkript komponens tartalmazza. Ezért töröljük ki az osztály definícióját! Az állomány többi része, egy kis módosítást leszámítva, vál-

zatlanul maradhat. (Természetes, ez csak azért lehetséges, mert a szkript komponensbe definiált tulajdonságokat és metódust, ugyanúgy neveztük el, mint a `CFKTransform` osztály megfelelő tulajdonságait, illetve metódusát.) A módosítást igénylő rész a következő:

```
Set objCFK = New CFKTransform
```

Jelen változatunkban, nem egy lapon belüli osztály objektumát kell létrehoznunk, hanem egy szkript komponensnek megfelelő osztályt. Ezért a fenti utasítást a következővel kell helyettesítenünk:

```
Set objCFK = Server.CreateObject("CFK.Scriptlet")
```

A `Server.CreateObject` metódus paramétere ugyanúgy definiálja a szkript komponenst, mint ahogy a Visual Basic program esetében láttuk: `CFK` játssza a könyvtár szerepét, az osztály neve mindig `Scriptlet`.

Ezzel el is készült a szkript komponenst használó ASP alkalmazás. Próbáljuk ki és győződjünk meg, hogy ugyanúgy működik, mint az előbbi megoldások. Most azonban erre az egyszerű fokátalakítási feladat megoldására egy szkript komponenssel rendelkezünk, amit szükség esetén más, amint láttuk, nem feltétlenül ASP alkalmazásba is használhatunk.

5.3 Visual Basic komponensek

Kisebb és egyszerűbb ASP komponensek előállítására a szkript komponensek és a Scriptlet Wizard egy jó és gyors megoldást képeznek. Bonyolultabb komponensek előállítására azonban más megoldást kell keressünk. Egy másik ok, amiért a fejlesztők csak gyors tervezésre használják a szkript komponenseket, a teljesen nyitott forráskód. Mint láttuk, a szkript komponensek forráskódja bármilyen ASCII szövegszerkesztővel megtekinthető. Ez lehetetlenné teszi a szkript komponensek használatát olyan szoftverben, melynek kódját, üzleti elgondolásokból védeni kell.

Ebben a szakaszban megvizsgáljuk, hogyan lehet a Visual Basic fejlesztőeszközzel ASP komponenseket előállítani. Az így létrehozott komponensek dinamikusan kötött könyvtárakban kapnak helyet, így bináris mivoltuk sokkal nehezebbé teszi a forráskódhoz való hozzáférést.

A Visual Basic fejlesztőeszközzel nagyon könnyen lehet ASP komponenseket előállítani. Visual C++ vagy más C/C++ fejlesztőeszközzel ellentétben, Visual Basic sok olyan előkészítő lépést, amit a COM komponensek előállítása megkövetel, elrejti a fejlesztő elől és automatikusan elvégzi. A COM komponensek Visual C++-szal való előállítása a COM technológia sokkal részletesebb ismeretét tételezi fel, mint ez a

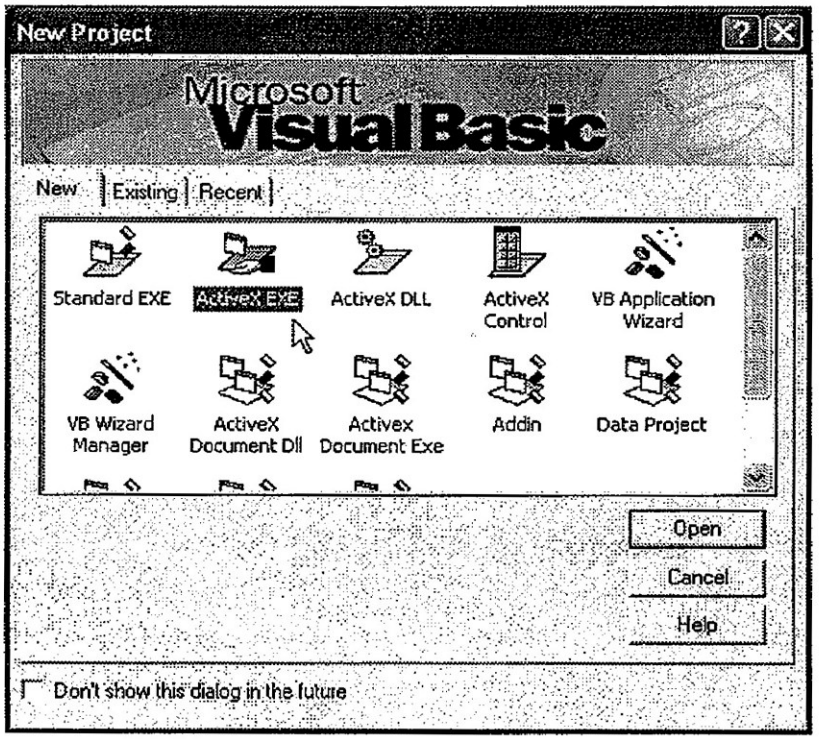
Visual Basic esetében szükséges. Ennek a könyvnek nem célja a COM technológia kimerítő bemutatása, így nem fogunk a COM komponensek C/C++ fejlesztőkörnyezetben történő előállításával foglalkozni, hanem a Visual Basic adta lehetőségekre korlátozzuk vizsgálatainkat. Miután a kedves Olvasó megértette a COM technológia lényegét, szükség esetén könnyebben alkalmazhatja a COM technológiáról szóló számos szakkönyvben található információt.

Legelőször megvizsgáljuk, hogyan állíthatunk elő Visual Basic-kel „semleges" komponenst, azaz olyan komponenst, melyet egyaránt használhatunk egy ASP és egy Win32 alkalmazásban. Ugyanazt a fokátalakítást elvégző komponenst fogjuk Visual Basic-kel létrehozni, mint amelyiket szkript komponensként is előállítottunk. Ez egyrészt megkönnyíti a Visual Basic forráskód követését azok részére is, akik eddig csak a VBScript szkriptnyelvet használták, másrészt lehetővé teszi, hogy a programozási nyelven túl, a DLL-ben foglalt komponensek létrehozásának a sajátosságaira koncentráljunk.

Miután tisztáztuk a COM komponensek létrehozását Visual Basic-ben, rátérünk az ASP specifikus komponensek előállítására.

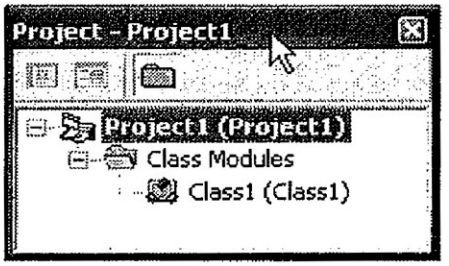
5.3.1 A Visual Basic komponensek előállítása

A DLL-be foglalt komponens elkészítésére a Visual Basic 6 verzióját fogjuk használni. Indítsuk el a fejlesztőeszközt és a megjelenő **New Project** ablakban válasszuk ki az **ActiveX DLL** projekttípust, majd nyomjuk meg az **Open** gombot (5.18. ábra)



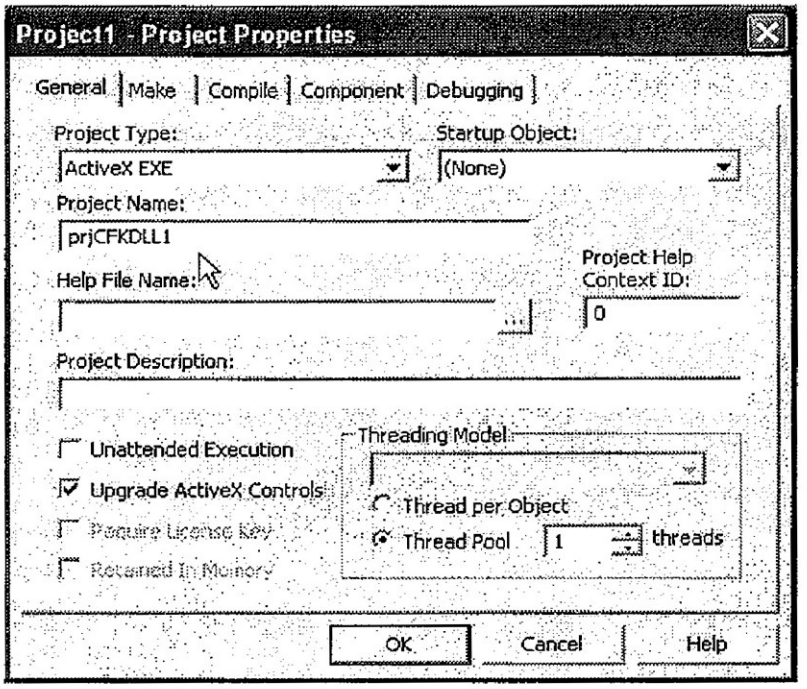
*5.18. ábra Létrehozunk egy **ActiveX DLL** típusú új projektet*

Visual Basic létrehoz egy projektet egy osztálymodullal. A projektet és a modult a projekttallózóban láthatjuk (5.19. ábra)



5.19. ábra A projekt és az osztálymodul a projekttallózóban

A projekt alapértelmezett neve `Project1`. Kattintsunk az egér jobb gombjával a projektre és a megjelenő környezetmenüből válasszuk ki a **Project1 Properties...** menüpontot. A megjelenő ablak több lapot tartalmaz. Hozzuk előre a **General** lapot és változtassuk meg a projekt nevét `prjCFKDLL1`-re. A többi beállításokat hagyjuk az alapértelmezett állapotukban, amint ezt az 5.20. ábrán láthatjuk.

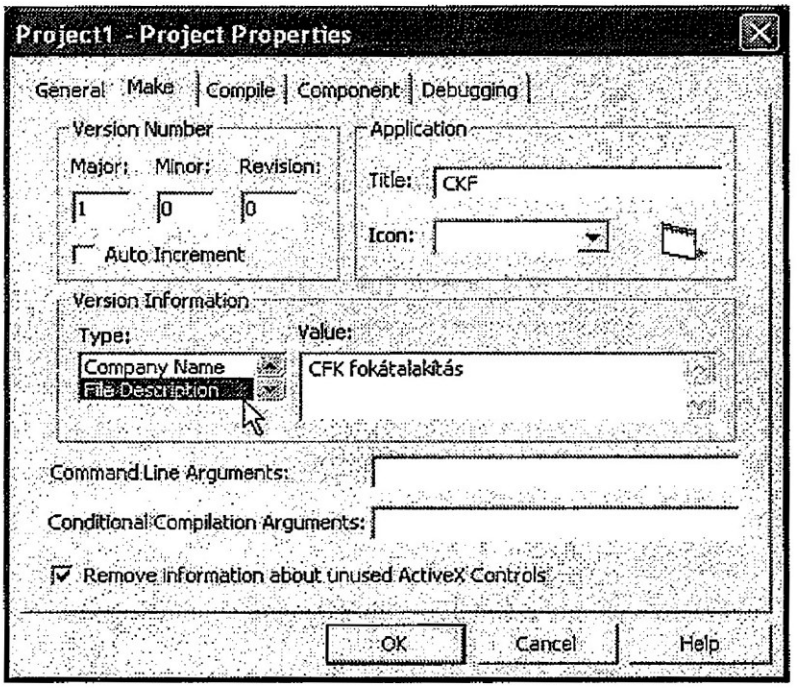


5.20. ábra Megváltoztatjuk a projekt nevét

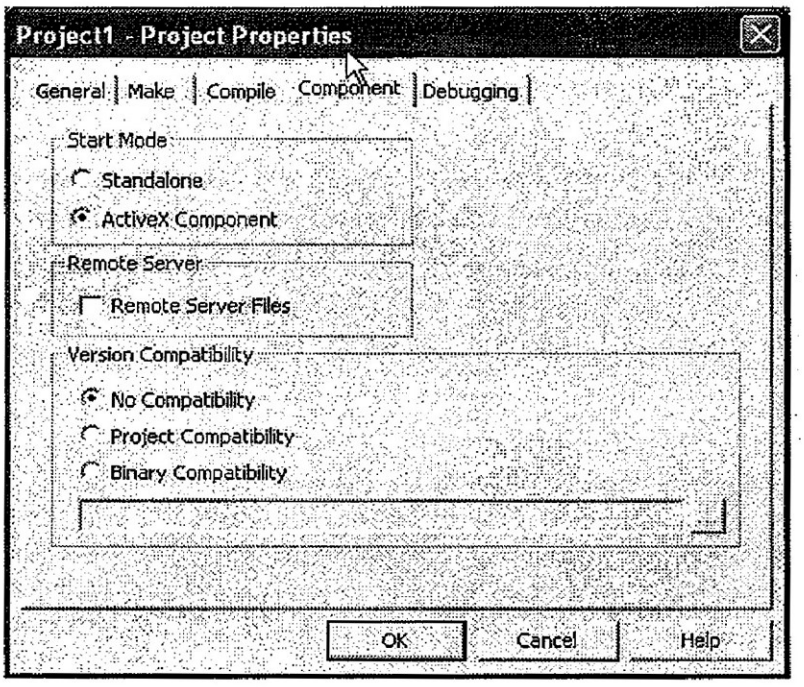
A **Make** lapon írjuk be az **Application** keretbe a projekt címét, a **Version Information** keretbe pedig írjuk be a projektet röviden leíró szöveget (**File Description**), úgy ahogyan ezt az 5.21. ábrán látjuk.

Hozzuk most a **Component** lapot előre! Itt fontos számunkra a **Version Compatibility** keret. Mint tudjuk a COM komponenseket rendszerünk a GUID, az általános egyértelmű azonosítóval tartja számon. Ha a **Version Compatibility** keretben a **No Compatibility** választógombot választjuk, akkor Visual Basic a projekt

minden egyes lefordításakor új GUID-ot ad a komponensnek. Ez azt jelenti, hogy ha a komponenst egy másik gépre másoljuk, minden újabb verzióval regisztrálni kell a komponenst. Ha a **Binary Compatibility** választógombot választjuk, akkor a Visual Basic nem gyárt új osztályazonosítót minden fordítás alkalmával. Egyelőre válasszuk ki a **No Compatibility** választógombot, később az első fordítás után kiválasztjuk a **Binary Compatibility** választógombot. A többi beállítást hagyjuk az alapértelmezett állapotban (5.22. ábra).



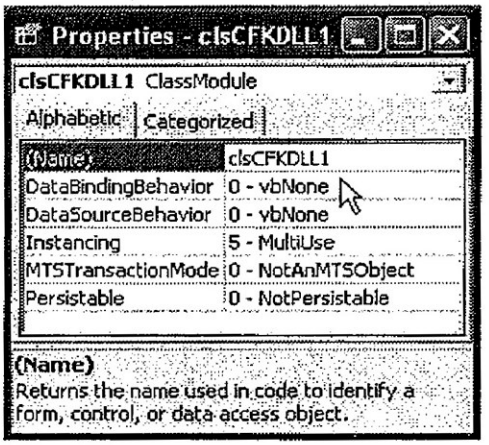
5.21. ábra Beadjuk a projekt címét és a leírását



5.22. ábra Beállítjuk a bináris kompatibilitást

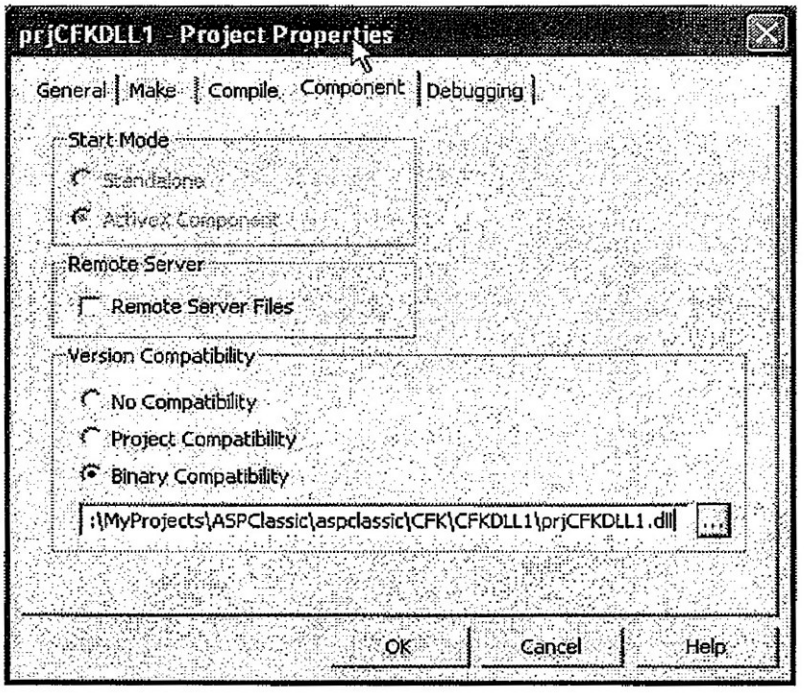
Ezek után nyomjuk meg az **OK** gombot, hogy rögzítsük a beállításokat.

Kattintsunk most az egér jobb gombjával a projekttallózóban a `Class1` osztálymodulra, amit a Visual Basic létrehozott. A környezetmenüből válasszuk ki a **Properties...** menüpontot és változtassuk meg a modul nevét `clsCFKDLL1`-re (5.23. ábra).



5.23. ábra Megváltoztatjuk az osztálymodul nevét

Egyenlőre a projektünk egy üres osztálymodulból áll, de előnyös, ha már most elmentjük egy mappába, úgy a projektet, mint a modult is. Szintén megkönnyíti a munkánkat, ha már most lefordítjuk az üres modult. A **File** menüből válasszuk ki a **Make prjCFKDLL1...** menüpontot és hozzunk létre egy üres DLL-t.



5.24. ábra Beállítjuk a bináris kompatibilitást

Most megváltoztathatjuk a kompatibilitási kikötéseket bináris kompatibilitásra. Ezért nyissuk újból ki a projekt tulajdonságait definiáló ablakot, hozzuk előre a **Component**

lapot és válasszuk ki a **Binary Compatibility** választógombot. Még mielőtt az **OK** gombra kattintanánk, meg kell adnunk a DLL-t, melyhez a bináris kompatibilitás viszonyítva lesz. (Ha nem adunk meg semmilyen DLL-t a viszonyításra, akkor egy hibajelentést kapunk.). Természetesen, a viszonyításra használt DLL a mi DLL-ünk, ezért kellett az üres projektet lefordítani. Írjuk be a `prjCFKDLL1.dll` elérési útját és nyomjuk meg az **OK** gombot (5.24. ábra).

Ezeket az előkészületeket elvégezve, elkezdhetjük a kód megírását. Mint láttuk, a projekt egy osztálymodulból áll. Ez a modul egy Visual Basic osztályt definiál. Osztályunk neve `clsCFKDLL1`. A `clsCFKDLL1` osztály ugyanazokkal a tulajdonságokkal és ugyanazzal a metódussal fog rendelkezni, mint a `CFK-WSH2_vbs.asp` állományban definiált `CFKTransform` VBScript osztály vagy a `CFK.sct` szkriptben definiált `CFKTransform` osztály.

Modulszinten három lokális változót definiálunk:

```
Private m_strFromDegree As String
Private m_strToDegree As String
Private m_dblDegree As Double
```

majd az ismerős `FormDegree`, `ToDergee`, és `Degree` tulajdonságot definiáljuk. Ezek közül csak az utolsót említem most meg:

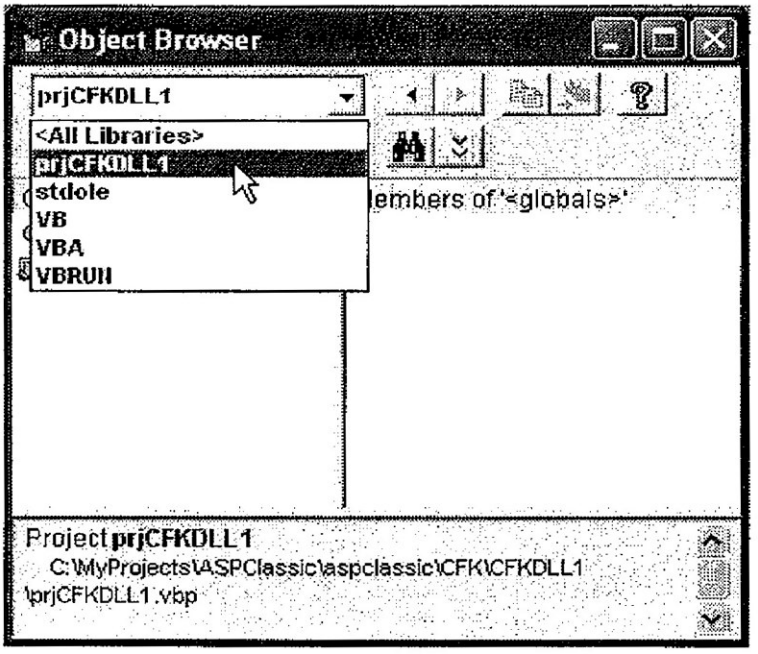
```
Public Property Let Degree(ByVal newData As Double)
  m_dblDegree = newData
End Property

Public Property Get Degree() As Double
  Degree = m_dblDegree
End Property
```

A `TransformDegree` függvény is alapjában a régi és csak a teljesség kedvéért ismétlem meg:

```
Public Function TransformDegree() As Double
  Select Case m_strFromDegree
    Case "Celsius"
      Select Case m_strToDegree
        Case "Celsius"
          TransformDegree = m_strToDegree
        Case "Fahrenheit"
          TransformDegree = (9/5) * m_dblDegree + 32
        Case "Kelvin"
          TransformDegree = m_dblDegree + 273.16
      End Select
```

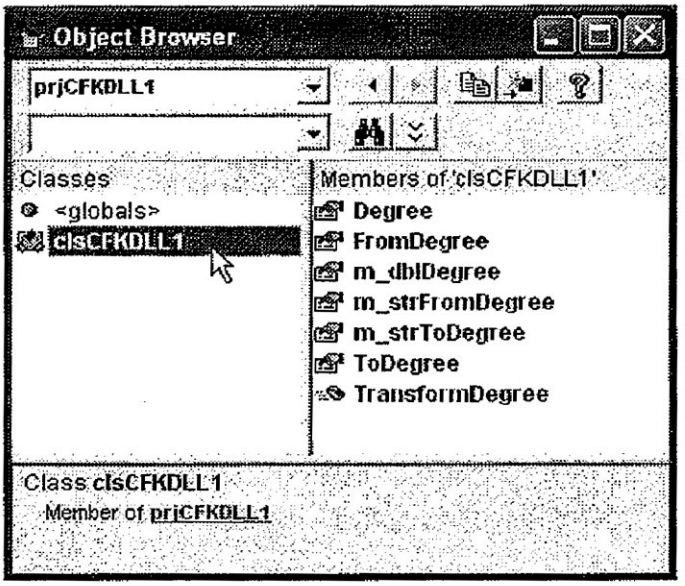
```
        Case "Fahrenheit"
          Select Case m_strToDegree
            Case "Celsius"
              TransformDegree = _
                          (5/9) * (m_dblDegree - 32)
            Case "Fahrenheit"
              TransformDegree = m_dblDegree
            Case "Kelvin"
              TransformDegree = _
                    (5/9) * (m_dblDegree - 32) + 273.16
          End Select
        Case "Kelvin"
          Select Case m_strToDegree
            Case "Celsius"
              TransformDegree = m_dblDegree - 273.16
            Case "Fahrenheit"
              TransformDegree = _
                    (9 / 5) * (m_dblDegree - 273.16) + 32
            Case "Kelvin"
              TransformDegree = m_dblDegree
          End Select
        End Select
    End Function
```

Nyissuk ki az **Object Browser**-t. Ezt a **View/Object Browser** menüponttal érjük el. Az **Objekt Browser** kombinált listájában megtaláljuk az előbb létrehozott komponenst (lásd az 5.25. ábrát).



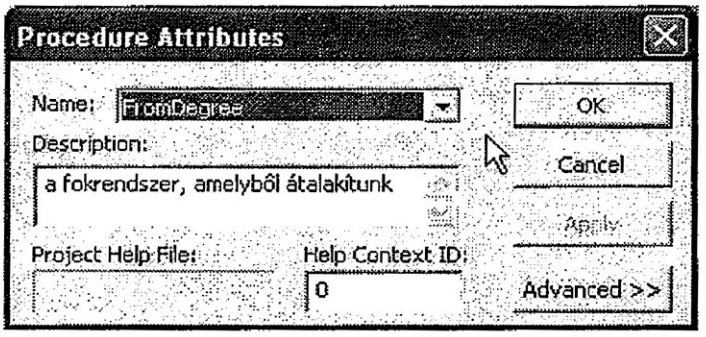
5.25. ábra A `prjCFKDLL1` *komponens az* **Object Browser** *listájában*

Ha rákattintunk a komponens `clsCFKDLL1` osztályára, akkor a jobb oldali listában látjuk a komponens tagjait (5.26. ábra).



5.26. ábra A `clsCFKDLL1` *osztály tagjai*

Ha rákattintunk az egér jobb gombjával a clsCFKDLL1 osztály egyik tagjára, akkor a környezetmenüből kiválaszthatjuk a **Properties...** menüpontot. A megjelenő ablakon magyarázatot fűzhetünk az osztály tagjaihoz (5.27. ábra)



5.27. ábra A `clsCFKDLL` *osztály tagjait elláthatjuk magyarázó szöveggel*

Fordítsuk le a projektet! Visual Basic létrehozza a `prjCFKDLL1.dll` komponenst. A projekt a letöltött ZIP állományban a `CFK\CFKDLL1` mappában található.

5.3.2 A Visual Basic komponens alkalmazása

Az előző alszakaszban létrehozott `prjCFKDLL1.dll` komponenst felhasználhatjuk más alkalmazásban. Mivelhogy egy „semleges" komponenst állítottunk elő, felhasználhatjuk egy Win32 és egy ASP alkalmazásban is. Vizsgáljuk meg először a Visual Basic alkalmazás esetét.

Az 5.1 szakaszban említettem, hogy egy Visual Basic programban a komponensnek megfelelő objektumot a

```
Set obj = CreateObject("ProjectName.ClassModulName")
```

utasítással hozzuk létre, ahol *ProjectName* a projekt neve és *ClassModulName* a komponenst definiáló osztályt tartalmazó modul neve.

Esetünkben ez az utasítás tehát a következő:

```
Set objCFK = CreateObject("prjCFKDLL1.clsCFKDLL1")
```

Hasonlítsuk össze ezt az utasítást a `prjCFKSCT.vbp` projektben használt utasítással:

```
Set objCFK = CreateObject("CFK.Scriptlet")
```

Mint látjuk, a hasonlóság nagy. A különbség a két komponens között legalább ilyen nagy. Nézzük meg közelebbről miben különbözik egy DLL komponens egy szkript komponenstől.

Ha ismét megvizsgáljuk az 5.2.6 alszakaszban bemutatott `prjCFKVBSCT.vbp` projekt `cmdTransform_Click` eseménykezelő eljárását, amelyik a fent idézett utasítást tartalmazza, akkor észrevehetjük, hogy az `objCFK` objektum deklarációjában az általános `Object` típust használtuk:

```
Dim objCFK As Object
```

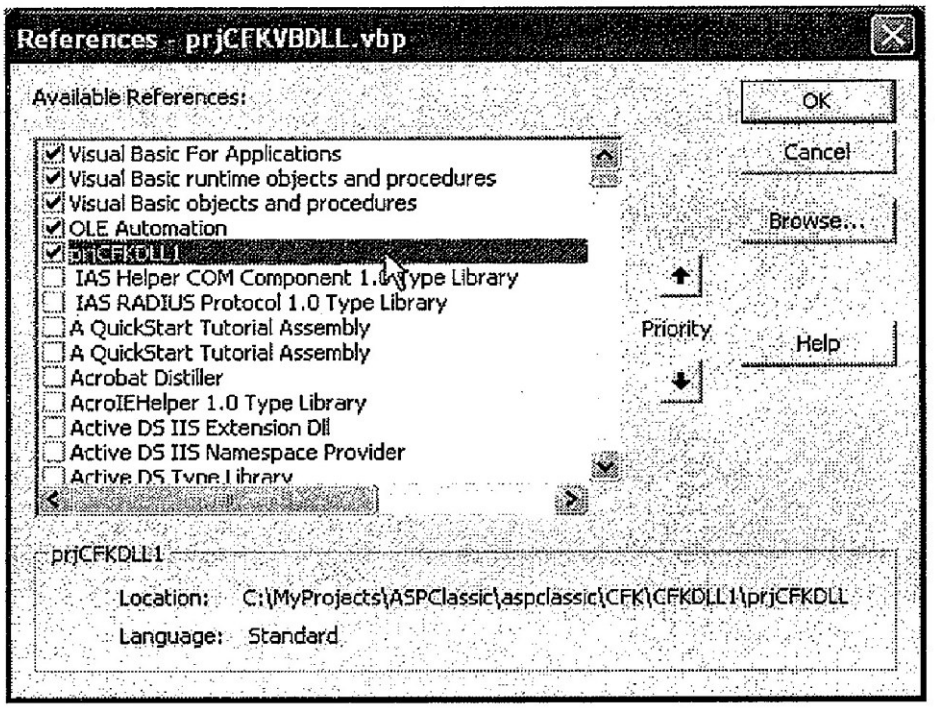
Erre azért volt szükség, mert projektünk fejlesztési időben nem vehetett tudomást a `CFK.Scriptlet` szkript komponensről. Csak lefordításával került a projektünk, a Windows **Registry** és a `scrobj.dll` komponens segítségével, a `CFK.Scriptlet` szkript komponenst definiáló `CFK.sct` állománnyal kapcsolatban (lásd az 5.9. ábrát).

Egészen más a helyzet egy DLL komponens esetében. Visual Basic regisztrálja a Windows **Registry**-ben a DLL komponenst. A regisztrálás a DLL-t előállító projekt lefordításakor történik. Ez azt jelenti, hogy a DLL komponens előállítása után máris felhasználhatjuk egy másik alkalmazásban.

Az 5.2.6 alszakaszban bemutatott Visual Basic alkalmazás projektje a `CFKVBSCT` mappába helyeztük el. Másoljuk most ennek a mappának a tartalmát a `CFKVBDLL` mappába, vagy nézzük meg a letöltött ZIP állományban az azonos nevű mappát. Nyissuk ki a `prjCFKVBSCT.vbp` projektet, változtassuk meg a nevét `prjCFKVBDLL`-re és mentsük el az új projektet a `prjCFKVBDLL.vbp` állományba. Ezután törölhetjük a mappából a `prjCFKVBSCT.vbp` állományt a `CFKVBSCT.exe` állománnyal együtt.

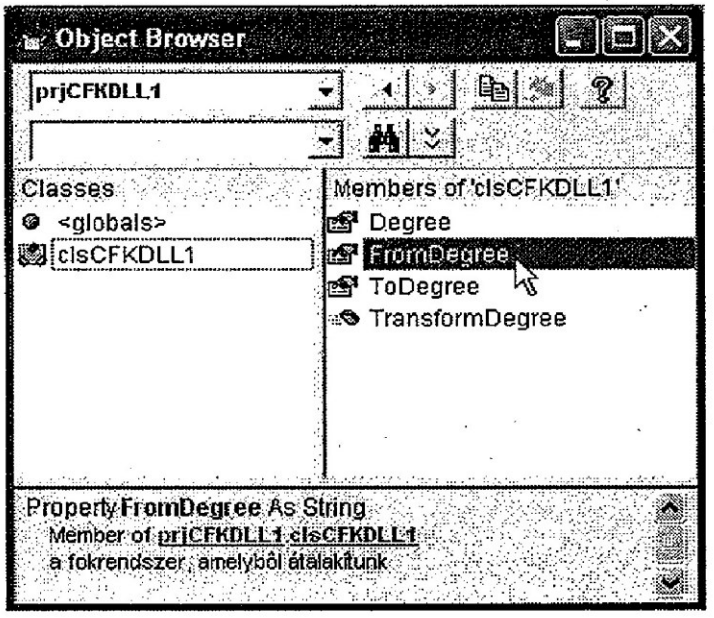
A Visual Basic-ben válasszuk ki a **Project/References...** menüpontot. A megjelenő ablak az ismert hivatkozásokat (*reference*) mutatja egy listában. A listában a `prjCFKDLL1` komponenst is megtaláljuk. Ha ez nem lenne így, akkor a **Browse...** pa-

rancsgombbal behozhatjuk a listába (lásd az 5.28. ábrát). A `prjCFKDLL1` komponens `prjCFKDLL1.dll` állományát a `CFKDLL1` mappába találjuk.



5.28. ábra A `prjDLL1` *komponens a hivatkozások listájában*

Nyissuk itt is az **Object Browser** ablakát. Itt is megtaláljuk a `prjDLL1` komponenst és a hozzátartozó `clsCFKDLL1` osztályt az osztálytagokkal. Ha az osztályra vagy az egyik tagra kattintunk, akkor a hozzáfűzött magyarázat is megjelenik (5.29. ábra).



5.29. ábra A `prjDLL1` *komponens és alkotórészei az* **Object Browser** *ablakban*

Nyissuk ki a `frmCFK` modul kódablakát. A kódot ismerjük, hiszen a `prjCFKVBSCT` projektből vettük át. Ha figyelmesen megvizsgáljuk a kódot, akkor mindjárt rájövünk, nagyon keveset kell rajta módosítani azért, hogy a `prjCFKDLL1` komponenst használhassuk.

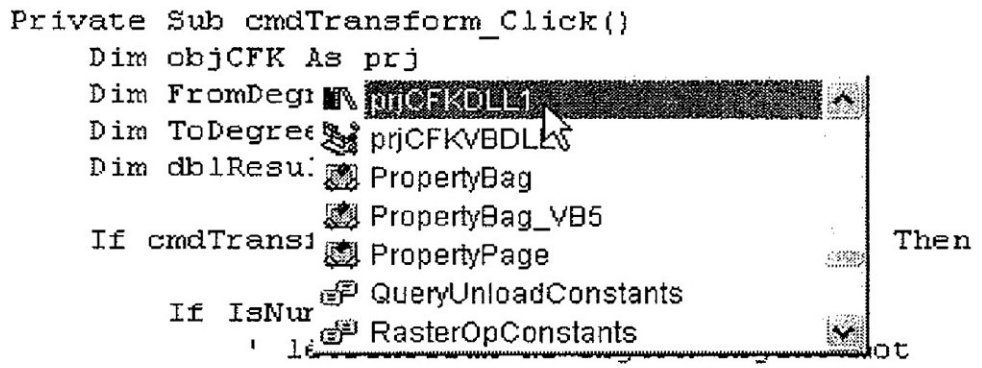
A módosításokat csak a `cmdTransform_Click` eljárásban kell elvégezni. Kezdjük el a

```
Dim objCFK As Object
```

utasítással. Mivelhogy Visual Basic már fejlesztési időben ismeri a `prjCFKDLL1` komponenst az általános objektumdeklaráció helyett, most megadhatjuk az objektum pontos típusát:

```
Dim objCFK As prjCFKDLL1.clsCFKDLL1
```

Még mielőtt beadnánk a típus teljes nevét, a Visual Basic segítségünkre siet (ha az **Auto List Members** opció a **Tools/Options.../Editor** ablakon nincs kikapcsolva), amint azt az 5.30. ábrán láthatjuk:

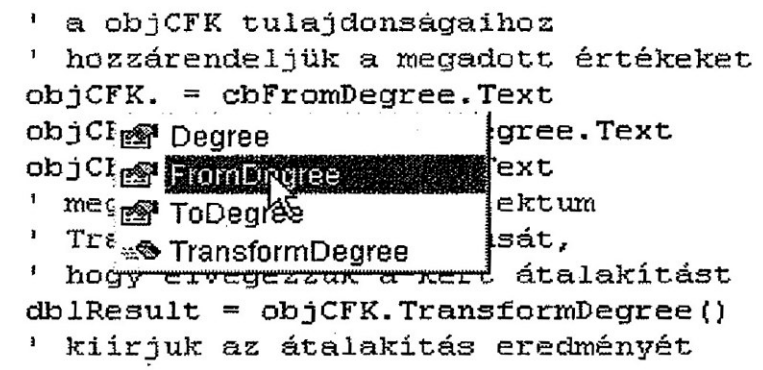


5.30. ábra A Visual Basic **Auto List Members** *opció felismeri a komponensünket*

Ezután a megszokott módon létrehozhatjuk az komponensünket megtestesítő objektumot:

```
Set objCFK = CreateObject("prjCFKDLL1.clsCFKDLL1")
```

Az objektum tulajdonságait is felismeri Visual Basic, amint ezt az 5.31. ábrán láthatjuk. Jelen esetben ez nem jelent különlegesen nagy segítséget, de egy bonyolultabb komponens esetében a tulajdonságok és metódusok megjelenítése az **Auto List Member** listában sokszor megkönnyíti és meggyorsítja a kód írását.



5.31. ábra A Visual Basic **Auto List Members** *opció felismeri a komponens tulajdonságait is*

Több átalakítást a kódban nem is kell eszközölnünk: projektünk készen áll a lefordításra. Fordítsuk le, és győződjünk meg, hogy ugyanúgy működik, mint a `prjCFKVBSCT` projekt.

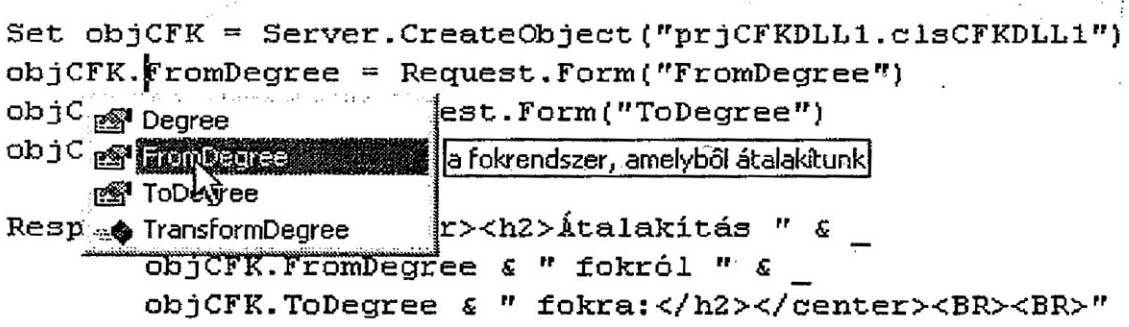
Ezek után a `prjCFLDLL1` komponens felhasználása egy ASP alkalmazásban már nem lehet nehéz. Hasonlóképpen, mint a Visual Basic alkalmazás esetében induljunk most is el a szkript komponenst használó alkalmazásból, azaz a `CFK-SCT_vbs.asp` ASP lapból. Készítsünk egy másolatot és adjunk a másolatnak a `CFK-DLL1_vbs.asp` nevet.

Nyilvánvaló, hogy az objektumot létrehozó utasítást kell módosítanunk:

```
Set objCFK = _
Server.CreateObject("prjCFKDLL1.clsCFKDLL1")
```

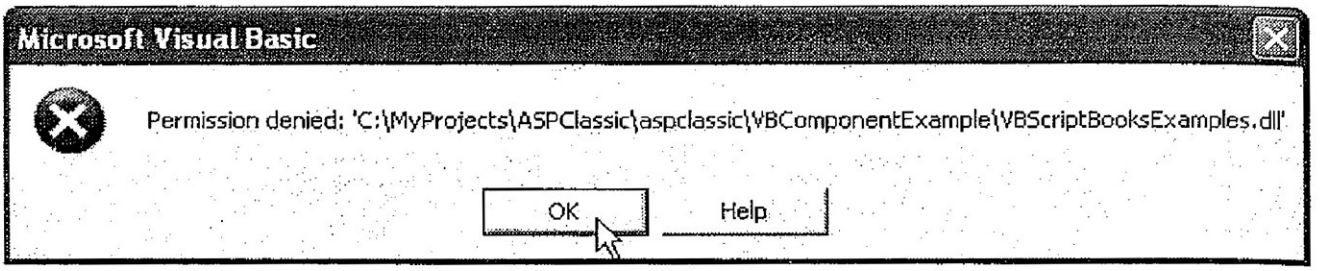
Ha az ASP állományt a Microsoft Visual InterDev 6.0 fejlesztőeszközzel írjuk, akkor ismét észrevehetjük, hogy a fejlesztőkörnyezet felismerte komponensünket (5.32. ábra).

Miután elvégeztük a módosítást és elmentettük az állományt, letölthetjük a böngészőnkbe, hogy meggyőződjünk helyes működéséről.



5.32. ábra Komponensünk a Microsoft Visual InterDev 6.0 fejlesztőeszközben

Bonyolultabb Visual Basic komponensek esetében, tanácsos a komponenst többször félkészállapotban egy ASP alkalmazásban kipróbálni, majd ismét folytatni a komponens fejlesztését a Visual Basic fejlesztőeszközzel. Ilyenkor, főleg Windows 2000 alatt gyakran előfordul, hogy a komponenst nem tudjuk újból lefordítani. Visual Basic ezt a tényt az 5.33. ábrán látható üzenettel hozza tudomásunkra.



5.33. ábra Visual Basic nem tudja a komponenst felordítani

Ha a komponensünket felhasználó ASP lapot letöltjük a böngészőnkben, akkor az IIS a komponensre való első utaláskor betölti a komponenst a memóriába és egy üzenetet küld a rendszernek, hogy a komponens le van foglalva. Ha közben egy program a komponens DLL állományát módosítani akarja, esetünkben Visual Basic akarja felülírni, akkor a rendszer ezt meggátolja és elküld a programnak egy **Permission denied** üzenetet. Ezt az üzenetet jeleníti meg Visual Basic.

Sajnos a komponens „felszabadítására" nem elegendő a webszerver leállítása az IIS konzolon belül. Ha az alkalmazásunkat tartalmazó webhelyre kattintunk az egér jobb gombjával és a környezetmenüből kiválasztjuk a **Stop** menüpontot, akkor a webszerver leáll, a memóriába töltött komponens viszont nem lesz felszabadítva. Hogy ez is bekövetkezzen, egy drasztikusabb módszerhez kell folyamodnunk: a webszervert a parancssoron (*command line*) beadott paranccsal kell leállítanunk.

Ha a fenti eset előfordul, akkor adjuk be a parancssoron a következő utasítást:

```
net stop iisadmin /y
```

A szolgáltatás elindítására adjuk be a

```
net start iisadmin
```

utasítást.

Befejezésül vizsgáljunk meg röviden még egy Visual Basic komponenst használó ASP alkalmazást.

A használt komponens szintén egy semleges, azaz környezetfüggetlen komponens lesz. Elvileg ASP alkalmazásokon kívül, más alkalmazásban is, például egy Visual

Basic alkalmazásban is felhasználható. Azonban mindjárt ki fog derülni, hogy annak ellenére, hogy ez elvileg lehetséges, mégsem ez a komponens „természetes" alkalmazása. A komponens ugyanis öt VBScript-tel foglalkozó könyv URL-jét, a könyv borítójának URL-jét és a könyv címét adja vissza véletlenszerű sorrendben. Az URL-ek az ismert www.amazon.com internetes könyvesbolt webhelyén vannak. Elképzelhető, hogy ASP alkalmazásokon kívül más alkalmazások is hasznát vehetik a komponenstől kapott webcímeknek, de nyilvánvaló, hogy elsődleges felhasználásuk egy ASP alkalmazásban a legmegfelelőbb.

A komponenst előállító Visual Basic projekt a letöltött ZIP állományban a `CFK\VBComponentExample` mappában található. A projekt a `VBScriptBooksExamples.dll` könyvtárat hozza létre, mely csak egy osztályt tartalmaz, a `VBScriptBooks` osztályt. Az osztály négy, csak olvasható tulajdonsággal rendelkezik:

```
Public Property Get Title(i As Integer) As String

Public Property Get Href(i As Integer) As String

Public Property Get Cover(i As Integer) As String
```

és

```
Public Property Get RandomBook() As Integer
```

Az első három tulajdonság a `i` rendszámú könyv címét, URL-jét, illetve a borítójának az URL-jét adja vissza. A negyedik tulajdonság véletlenszerűen egy 1 és 5 között levő egész számot ad vissza.

A komponenst felhasználó ASP alkalmazás a jelen fejezet megfelelő mappájában található és a `VBComponentExample_vbs.asp` nevet viseli.

5.4 Visual Basic ASP komponensek

A mostanáig megvizsgált Visual Basic komponensek úgynevezett „semleges" komponensek voltak. Ezeket a komponenseket környezetfüggetlen vagy kontextusfüggetlen komponenseknek is nevezzük, ezzel az elnevezéssel is aláhúzva azon tulajdonságukat, hogy alkalmazásuk a környezettől független: egyaránt alkalmazhatjuk ASP vagy más alkalmazásban.

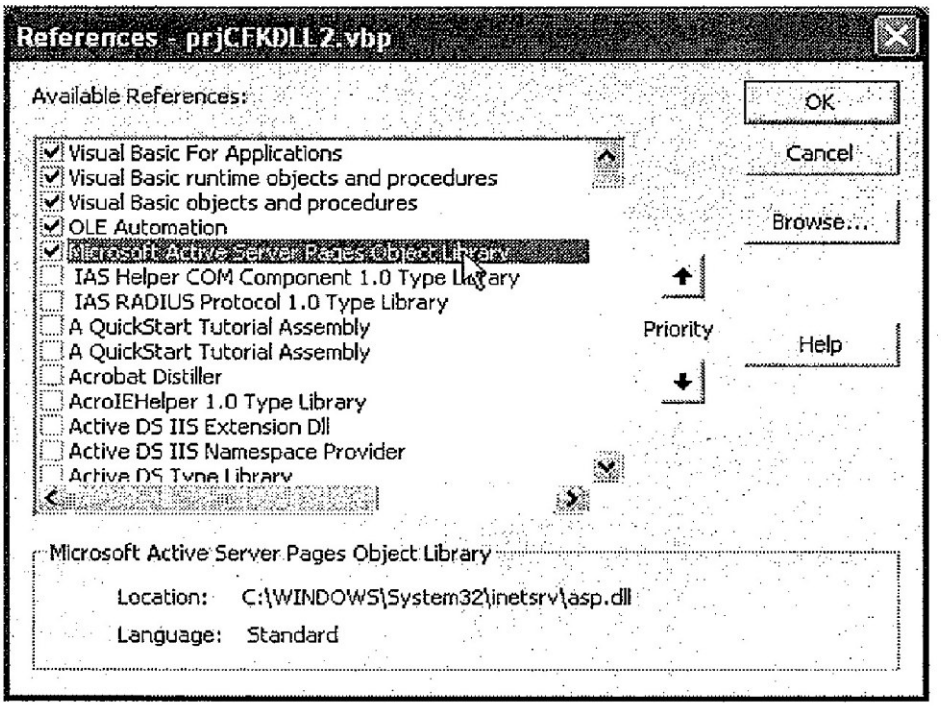
Most eljött az idő, hogy megvizsgáljuk az ASP specifikus komponenseket. Amint látni fogjuk, megértésükre már sok mindent megismertünk és kevés új fogalmat kell még tisztázni, hogy előállíthassuk első Visual Basic ASP komponensünket.

Az ASP specifikus komponensek abban térnek el a környezetsemleges komponensektől, hogy az előbbiek ASP objektumokat használnak, mint a `Response` vagy `Request` vagy más ASP-sajátos objektumok

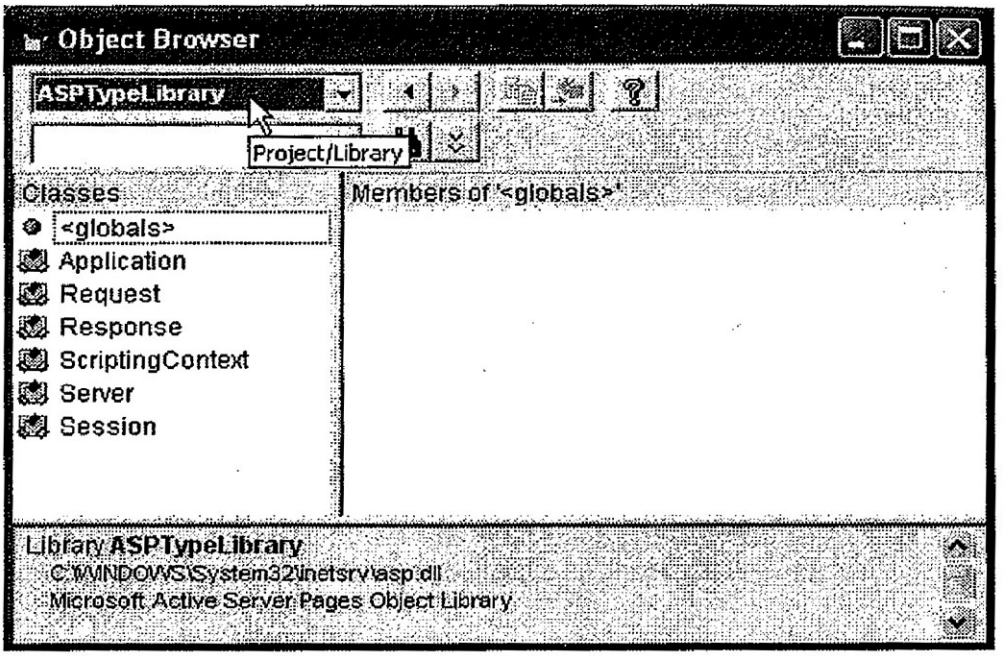
Máris felvetődik a kérdés: honnan „ismeri" az ASP specifikus komponens az ASP objektumokat? Ezzel a kérdéssel máris rátapintottunk az ASP komponensek egyik lényegére. A választ az ASP komponensek előállítása folyamán fogjuk megkapni.

Példaként, alakítsuk át az immár jól ismert fokátalakítási komponenst egy ASP komponensre. Indítsuk el Visual Basic-ot és nyissunk ki egy új ActiveX DLL típusú projektet. Nevezzük el a projektet `prjCFKDLL2`-nek, az osztálymodult pedig `clsCFKDLL2`-nek. Ezután vegyük fel a projektbe a Microsoft Active Server Pages Object Library dinamikusan kötött könyvtárat (5.34. ábra). Ha nem találjuk a **Project/Referencess...** menüponttal megjelenített listában, akkor tallózunk a `Windows\System32\inetsrv\asp.dll` állományhoz, amelyik a könyvtárat tartalmazza. Ez a könyvtár nyitja meg az ASP objektumok világát a komponensünk számára.

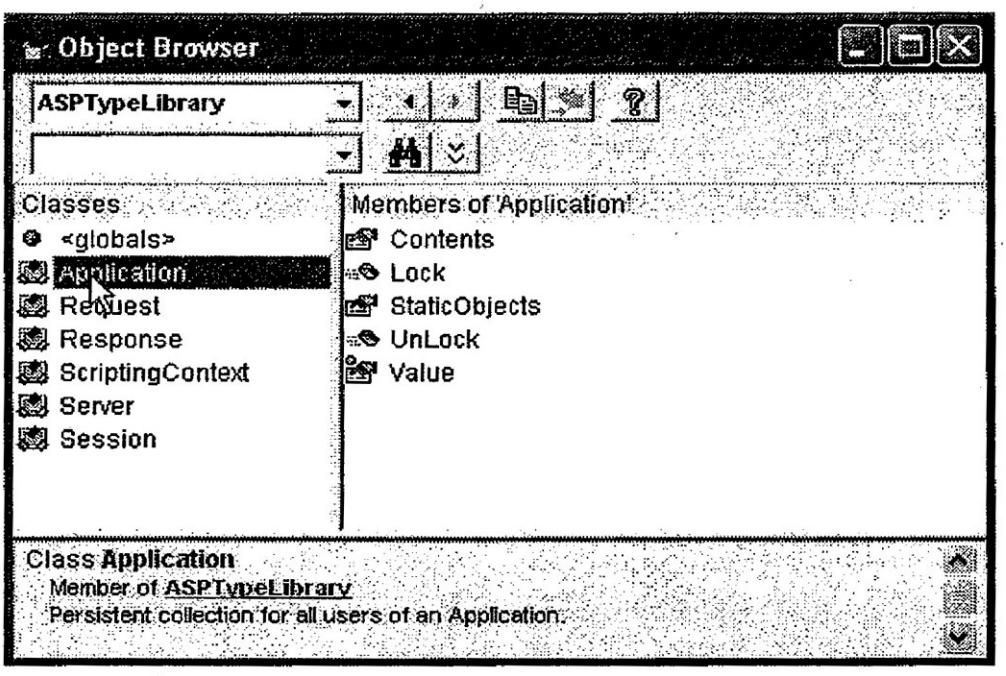
Nyissuk ki az **Object Browser** ablakát és válasszuk ki a kombinált listából az **ASPTypeLibrary** bejegyzést. Az **Object Browser** megmutatja a könyvtárban létező objektumokat (5.35. ábra). Egy kivételével mindegyiknek ismerősnek kell lenni. Ha az egyik ismert objektumra, például az `Application` objektumra kattintunk, akkor az ablak jobb oldali részében láthatjuk az objektum tulajdonságait és metódusait (5.36. ábra).



5.34. ábra Bekötjük a Microsoft Active Server Pages Object Library könyvtárat a projektbe

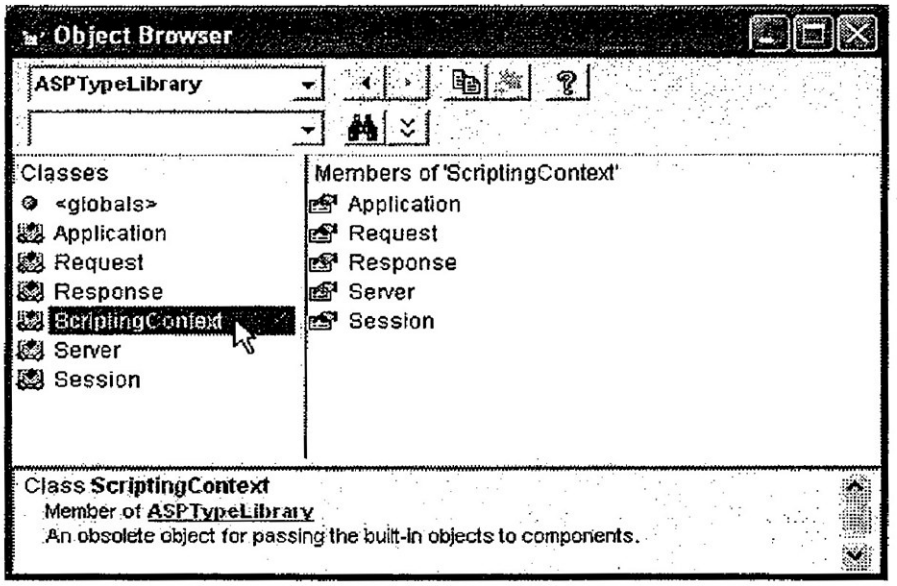


5.35. ábra Az ASP belső objektumai az **Object Browser** *ablakában*



5.36. ábra Az `Application` *objektum tulajdonságai és metódusai*

Az ismeretlen objektum a `ScriptingContext` objektum. Kattintsunk erre az objektumra, hogy megnézzük, mit mutat róla az **Object Browser** (5.37. ábra).



5.37. ábra A `ScriptingContext` *objektum*

A `ScriptingContext` objektum öt tulajdonsággal rendelkezik és nincs egyetlen metódusa sem. A tulajdonságok az ismert ASP objektumok nevét viselik. Az objektumot kísérő magyarázat (*An obsolete object for passing the built-in objects to components*) szerint egy felesleges objektumról van szó, mely arra szolgál, hogy az ASP belső objektumait hozzáférhetővé tegye komponensek részére. A `ScriptingContext` objektum természetes csak az ASP szempontjából felesleges, a komponensek szempontjából igenis nagyon hasznos és szükséges. Ha az egyébként csak olvasható tulajdonságokra kattintunk, akkor a hozzáfűzött magyarázatból megtudjuk, hogy ezek a tulajdonságok az azonos nevű ASP objektumot adják vissza.

Hogy használjuk fel a `ScriptingContext` objektumot a komponensünket előállító projektbe? Mivelhogy a `ScriptingContext` objektumot tartalmazó könyvtárat bekötöttük a projektünkbe, semmi akadálya annak, hogy egy `ScriptingContext` típusú objektumot deklaráljunk. Ugyanezt megtehetjük a többi objektumtípussal is, amelyeket a Microsoft Active Server Pages Object Library könyvtár definiál. Deklaráljuk tehát modulszinten a következő objektumokat:

```
Private MyScriptingContext As ScriptingContext
Private MyServer As Server
Private MyApplication As Application
Private MySession As Session
Private MyRequest As Request
Private MyResponse As Response
```

Természetesen csak azokat az objektumokat kell deklarálnunk, amelyeket használni akarunk. Mivelhogy ezeket az objektumváltozókat modulszinten deklaráltuk, bárhol a modulon belül szabadon használhatjuk. Egyelőre ezeket az objektumokat csak deklaráltuk, azaz helyet foglaltunk le számukra a memóriában, de még nem töltöttük meg tartalommal. Erre nemsokára sor kerül.

Miután elkészültünk a komponensünkkel, úgy mint mostanáig mindig, az ASP alkalmazásban a `Server` objektum `CreatObject` metódusával létrehozhatjuk a komponenst megtestesítő objektumot:

```
Dim objCFK
Set objCFK = Server.CreateObject("prjCFKDLL2.clsCFKDLL2")
```

Itt álljunk meg egy pillanatra, hogy kihangsúlyozzam: *a* `Server` *objektum* `CreateObject` *metódusát kell meghívnunk, nem a VBScript* `CreateObject` *utasítását!*

Ha a `Server` objektum `CreateObject` metódusát hívjuk meg, akkor ASP létrehozza a komponensnek megfelelő objektumot, és máris megpróbálja a komponens `OnStartPage` metódusát meghívni. Ha viszont „csak" a VBScript `CreateObject` utasítását hívjuk meg, akkor ASP nem kísérli meg a komponens `OnStartPage` metódusát meghívni. Pontosabban fogalmazva: minden komponens esetében ASP feltételezi, hogy a komponens rendelkezik egy `OnStartPage` és egy `OnEndPage` metódussal.

Az `OnEndPage` metódusnak nincs paramétere, az `OnStartPage` deklarációja a következő

```
Public Sub OnStartPage( _
        PassedScriptingContext As ScriptingContext _
)
```

Mint látjuk, ebben az eljárásban juttatja el az ASP a `ScriptingContext` objektumot a komponensünkhöz. Ez a legjobb hely, ahol átvehetjük az ASP belső objektumait, hogy azután szabadon használhassuk a komponensünkben:

```
' közvetlenül a komponens létrehozása után
' lesz végrehajtva
Public Sub OnStartPage( _
    PassedScriptingContext As ScriptingContext _
)

    Set MyScriptingContext = PassedScriptingContext

    With MyScriptingContext
        Set MyServer = .Server
        Set MyApplication = .Application
        Set MySession = .Session
        Set MyRequest = .Request
        Set MyResponse = .Response
    End With

End Sub
```

Az OnEndPage eljárást az ASP közvetlenül a komponens megsemmisítése előtt hívja meg. Ez vagy akkor történik, amikor a komponenst felhasználó ASP lap a

```
Set objCFK = Nothing
```

utasítást hajtja végre vagy amikor az alkalmazásunk véget ért. Következésképpen az OnEndPage eljárás a legjobb hely, ahol a memóriát felszabadíthatjuk:

```
' közvetlenül a komponens megsemmisítése előtt
' lesz végrehajtva
Public Sub OnEndPage()
    Set MyScriptingContext = Nothing
    Set MyServer = Nothing
    Set MyApplication = Nothing
    Set MySession = Nothing
    Set MyRequest = Nothing
    Set MyResponse = Nothing
End Sub
```

Mielőtt tovább mennénk, jegyezzük meg, hogy az OnStartPage és OnEndPage eljárások a készülő komponens metódusai. Minden ASP specifikus komponensnek rendelkeznie kell ezzel a két nyilvános (*public*) metódussal. A metódusokat feltétlenül a **Public** módosítóval kell definiálnunk, különben ASP nem tudja végrehajtani.

A készülő komponens az ismert fokátalakitási feladatot oldja meg. Előbbi változataiból tudjuk, hogy az osztálynak három privát változója:

```
Private m_strFromDegree As String
Private m_strToDegree As String
Private m_dblDegree As Double
```

három tulajdonsága: FromDegree, ToDegree és Degree és egy metódusa: TransformDegree van. A tulajdonságok és a metódus definíciója azonos a prjCFKDLL1 komponens esetében használt definícióval, azzal az egy különbséggel, hogy a jelen esetben elegendő, ha a tulajdonságokat és a metódust a **Private** módosítóval definiáljuk. Ugyanis ezeket nem fogjuk az osztályon kívül meghívni.

Ezzel szemben egy újabb metódust definiálunk a komponensünk részére, melyet viszont a komponens felkínál a kliensalkalmazásnak. A **Public Sub** SetUp() metódusról van szó, melynek forráskódját itt láthatjuk:

```
Public Sub SetUp()

  If MyScriptingContext Is Nothing Then
    MsgBox "Ezt a komponenst csak egy ASP " _
           "alkalmazáson belül lehet használni", _
            vbCritical
```

```
    Exit Sub
  End If

  If MyRequest.Form("FromDegree") <> "" And _
    MyRequest.Form("ToDegree") <> "" And _
    MyRequest.Form("deg") <> "" Then
    If IsNumeric(MyRequest.Form("deg")) Then
      FromDegree = MyRequest.Form("FromDegree")
      ToDegree = MyRequest.Form("ToDegree")
      Degree = MyRequest.Form("deg")

      MyResponse.Write ("<center><h2>Átalakítás " & _
      FromDegree & " fokról " & _
      ToDegree & " fokra:</h2></center><BR><BR>")

      MyResponse.Write ("<h3>" & ToDegree & "º " & _
      FromDegree & " foknak megfelel " & _
      TransformDegree & "º " & _
      ToDegree & " fok." & "</h3><BR><BR>")

      MyResponse.Write ("<HR><A HREF=""" & _
      MyRequest.ServerVariables("SCRIPT_NAME") & """>Vissza</A>")
    Else
      MyResponse.Write ("<CENTER>" & _
      "<H2><FONT color=red> Hibás foszám!" & _
      "</FONT></H2></CENTER>")

      MyResponse.Write ("<HR><A HREF=""" & _
      MyRequest.ServerVariables("SCRIPT_NAME") & """>Vissza</A>")
    End If
    MyResponse.End
    Exit Sub
  End If

  MyResponse.Write ("<CENTER><H2>" & _
  "Celsius-Fahrenheit-Kelvin fokátalakítás. " & _
  "</H2><BR><BR>")

  MyResponse.Write ("<FORM ACTION=""")
  MyResponse.Write ( _
  MyRequest.ServerVariables("SCRIPT_NAME") & """")
  MyResponse.Write ( _
  " METHOD=""POST"" id=form1 name=form1>")

  MyResponse.Write ("<P>")

  MyResponse.Write ("<SELECT id=""FromDegree"" " & _
  "name=""FromDegree"" style=""HEIGHT: 25px; " & _
  "WIDTH: 100px"">")
  MyResponse.Write ("<OPTION selected " & _
  "value=""Celsius"" >Celsius</OPTION>")
  MyResponse.Write ("<OPTION value=" & _
  """Fahrenheit""> Fahrenheit</OPTION>")
```

```
MyResponse.Write ("<OPTION value=" & _
"""Kelvin"">Kelvin</OPTION>")
MyResponse.Write (" </SELECT>")
MyResponse.Write (" fokról")
MyResponse.Write ("<INPUT id=""deg"" " & _
"name=""deg"" size=5 value=0> fokot")

MyResponse.Write ("<SELECT id=""ToDegree"" " & _
"name=""ToDegree"" style=""HEIGHT: 25px; " & _
"WIDTH: 100px"">")
MyResponse.Write ("<OPTION selected value=" & _
"""Celsius"" >Celsius</OPTION>")
MyResponse.Write ("<OPTION value=" & _
"""Fahrenheit"">Fahrenheit</OPTION>")
MyResponse.Write ("<OPTION value=" & _
"""Kelvin"">Kelvin</OPTION>")
MyResponse.Write (" </SELECT>")
MyResponse.Write (" fokra")
MyResponse.Write (" <INPUT type=""submit"" " & _
"value=""átalakítani!"" id=submit1 name=submit1>")
MyResponse.Write ("</P>")
MyResponse.Write ("</FORM>")
MyResponse.Write ("</CENTER>")

End Sub
```

Vizsgáljuk meg ezt a metódust! Mindjárt az elején ellenőrizzük, hogy rendelkezünk-e egy `ScriptingContext` objektummal. Az objektumot az `OnStartPage` metódusban továbbítja az ASP a komponensnek. Az `OnStartPage` metódust azonban az ASP csak akkor hívja meg, ha az objektumot a `Server.CreateObject` metódussal hozzuk létre. Ha nem rendelkezünk a `ScriptingContext` objektummal, akkor nyilvánvaló, hogy az objektum nem a `Server.CreateObject` metódussal lett létrehozva. Következésképpen, tudatjuk a felhasználót, hogy ezt a komponenst csakis ASP alkalmazások használhatják és elhagyjuk a `SetUp` metódust.

A hátramaradt forráskód ismerős lesz. Ez a kódrészlet alapvetően megegyezik az ismert `CDF-DLL1_vbs.asp` ASP lap tartalmával. Ez a lap ASP és HTML kódot tartalmaz.

Az ASP kódot majdnem egészében megtaláljuk a `SetUp` metódus kódjában is. Ez eltérés egyrészt abban áll, hogy természetesen most nem kell a komponenst létrehoznunk azért, hogy tulajdonságait és metódusát használhassuk, hiszen egyelőre még a komponensen „belül" vagyunk. Másrészt, az ASP objektumokat csak a komponens belül definiált változókkal tudjuk elérni.

A `CDF-DLL1_vbs.asp` ASP lap HTML kódját is megtaláljuk a `SetUp` metódus kódjában, azzal a különbséggel, hogy most a `Response` objektum `Write` metódusával íratjuk a kliensalkalmazás böngészőjébe.

Mit tartalmaz akkor a komponenst felhasználó ASP lap? Ennek a lapnak a forráskódja nagyon egyszerű:

```
Dim objCFK
Set objCFK = _
Server.CreateObject("prjCFKDLL2.clsCFKDLL2") objCFK.Setup
Set objCFK = Nothing
```

Fordítsuk le a komponenst és vizsgáljuk meg működését. A jelen fejezetnek megfelelő mappába helyeztem el a `CFK-DLL2_vbs.asp` lapot, amivel ezt megtehetjük.

6. Jelszóval védett weblapok - egy példaalkalmazás előállítása

Megérkeztünk a könyv utolsó fejezetéhez. Itt az idő, hogy megismerkedjünk egy terjedelmesebb és összetettebb webalkalmazással. Ebben a fejezetben ezt fogjuk tenni.

Bizonyára mindegyikünk találkozott olyan webhellyel, ahol bizonyos lapokat csak regisztrált látogatók tekinthettek meg. Az angol nyelvű szakirodalom ilyen webterületeket *members area*-nak nevezi. Egy ilyen webalkalmazást fogunk ebben a fejezetben megismerni.

Aligha létezhet egy komolyabb webalkalmazás adatbázis nélkül. Ha egy alkalmazás számon akarja tartani látogatóit, meg akarja látogatóit egymástól különböztetni, esetleg a látogatók egyéni beállításait kívánja megőrizni és szükség esetén visszaállítani, akkor egy adatbázis használata elkerülhetetlen.

A most bemutatásra kerülő alkalmazás is adatbázist használ. Ez a könyv nem foglakozik az adatbázis-kezeléssel, az alkalmazás megértéséhez viszont legalább az adatbáziskezelés alapjait kell az Olvasónak ismernie. Egy jó és gyors magyar nyelvű bevezetőt az adatkezelésről az ActiveX Data Object (ADO) objektummodellel a következő cím alatt található: http://www.prog.hu/article.php?aid=373

Az alkalmazásban fel fogunk használni, ha nem is mindent, de sok mindent, amit az előző fejezetekben megismertünk. Ezen kívül látni fogjuk a 2. fejezetben megígért `Request.TotalBytes` tulajdonság, a `Request.BinaryRead` metódus és a `<INPUT type="file">` vezérlő alkalmazását.

Az alkalmazás teljes forráskódja a letöltött ZIP állományban a `membersarea` mappában található.

6.1 A feladat meghatározása és előkészületek

Alkalmazásunk a következőket biztosítja:

- új tagok regisztrálása;
- regisztrált tagok bejelentkezése a rendszerbe jelszó alapján;
- elfelejtett jelszó elküldése;
- regisztrált tagok adatainak módosítása;
- tagok törlése az adatbázisból;

- egy kiemelt tag, a rendszergazda, bejelentkezése különleges, csak számára fenntartott, műveletek elvégzésére;
- hírlevelek elküldése a tagoknak (csak a rendszergazda jogosult);
- a tagok adatainak módosítása vagy törlése (csak a rendszergazda jogosult).

Az alkalmazást képező állományokat egy `membersarea` nevű mappába helyezzük. Ez lesz alkalmazásunk törzsmappája. Ezen belül még három almappára lesz szükségünk: `dbase`, `protected`, és `utils`.

A `utils` almappa az alkalmazást képező állományokat tartalmazza. A védett állományokat a `protected` almappába helyezzük el, míg a `dbase` almappába az adatbázisunk kerül. Az egyszerűség kedvéért egy MS Access 2002 adatbázist használtam. MS SQL vagy Oracle adatbázis szerintem egy ilyen példaalkalmazás esetében felesleges, de adott esetben kevés módosítással ezeket az adatbázisokat is alkalmazhatjuk.

Adatbázisunk `membersdb.mdb` és egyetlen egy `members` táblája van. (Ugyanebben a mappában található a `membersdb97.mdb` és a `membersdb2000.mdb` MS Access 97, illetve 2000 adatbázis, amit az eredeti helyett használhatunk, miután nevét megfelelőképpen megváltoztattuk.) Az adatmezőket és az adattípusokat a következő táblázatban láthatjuk:

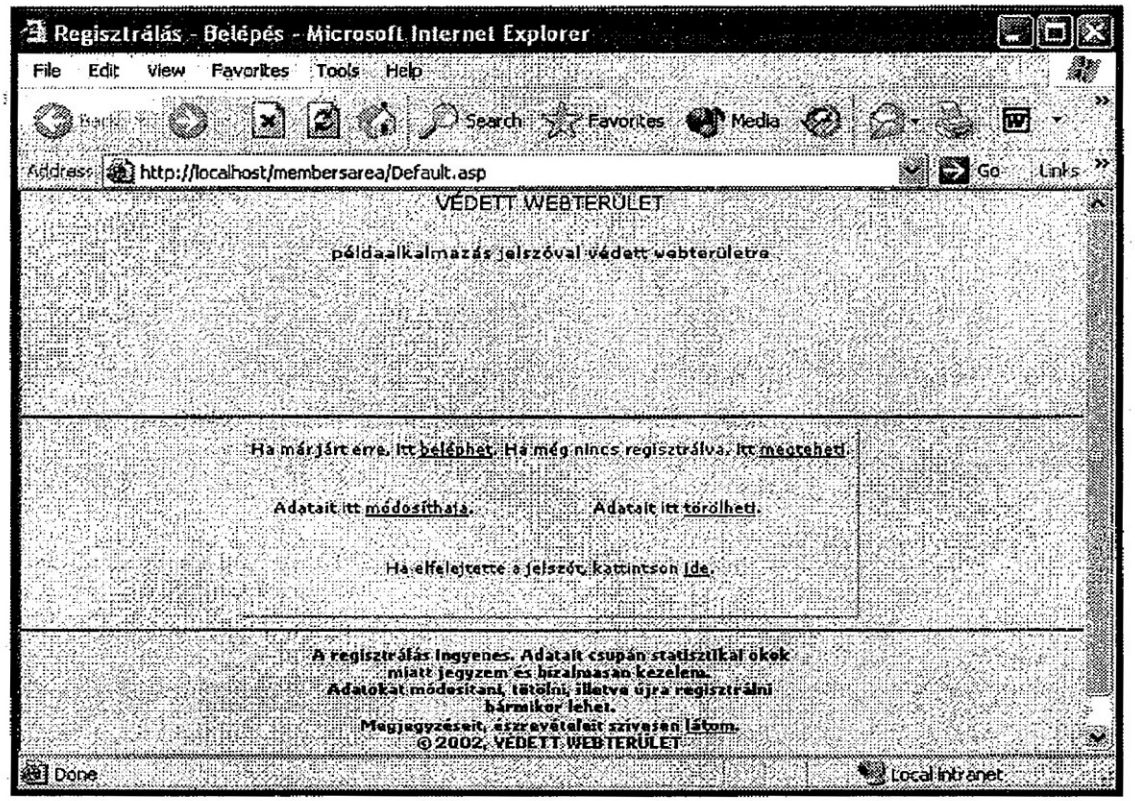
adatmező	adattípus	magyarázat
UID	AutoNumber	felhasználó azonosító
FirstName	Text	keresztnév
SecondName	Text	vezetéknév
Email	Text	email cím
Password	Text	jelszó
RegisteredOn	Date/Time	regisztráció időpontja
IsAdmin	Yes/No	a rendszergazda estében Yes, különben No értékű

A `members` tábla legalább egy, a rendszergazda adatainak megfelelő rekordot kell tartalmaznia. Az **Email**, **Password** adatmezők kitöltése kötelező. Az **IsAdmin** mezőben levő jelölőnégyzetet csak a rendszergazdánál szabad és kell kijelölni. A példaalkalmazás esetében a rendszergazda adatai a következők. **Email**:admin@webhelyem.net, **Password**: engedjbe.

Ezzel az előkészületeket befejeztük és elkezdhetjük a forráskód írását, illetve elemzését.

6.2 A kezdő és a regisztráló lap

Alkalmazásunk kezdőlapja a `default.asp` lap, amit a törzsmappába helyezzünk el. A kezdőlapot a 6.1. ábrán láthatjuk.



6.1. ábra Az alkalmazás kezdőlapja

Forráskódjából láthatjuk, hogy egy egyszerű HTML kódot tartalmazó lapról van szó, kevés ASP elemmel. Mindjárt a lap elején a `Session.Abandon` utasítással lezárjuk az aktuális munkamenetet (*session*), töröljük az összes munkamenetszintű (*session-level*) változót, így az alkalmazást egy vadonatúj munkamenettel kezdjük. Erre azért van szükség, mert később munkamenetszintű változókba fogunk információt az aktuális felhasználóról tárolni, tehát az alkalmazás elején "tiszta lappal" kell indulnunk. Ezen kívül, beillesztjük az állományba a `utils/data.asp` állományt, amelyben az alkalmazás folyamán előforduló "globális" változókat definiáljuk:

```
<!--      #include file="utils/data.asp"          -->
```

A `utils/data.asp` állományban megadott változók módosításával konfigurálhatjuk az alkalmazást. Ezekre a változókra később még visszatérek. Egyelőre csak a

```
site_title="VÉDETT WEBTERÜLET"
editors_email="editor@webhelyem.net"
```

változók érdekelnek, mert értéküket a kezdőlapon megjelenítjük. Az `default.asp` állomány fejlécében beillesztjük a `utils/cssrules.asp` állományt. Szerepe csak annyiban áll, hogy beilleszti a `utils/cssrules1.css` stíluslapot:

```
<!-------cssrules.asp------------------->
<!--    #include file="cssrules1.css" -->
```

Azért választottam ezt a megoldást, mert így a használt stíluslapot könnyen cserélhetjük: a `utils/cssrules.asp` állományt mindegyik, stíluslapot használó állományba beillesztjük, és ha egy másik stíluslapot akarunk használni, akkor csak a `utils/cssrules.asp` állományban kell a változást végrehajtani. A `utils` mappába öt stíluslap található: `ccsrules1.css`, ..., `cssrules5.css`.

A kezdőlap lehetőséget nyújt a látogatónak, hogy beregisztrálja magát, hogy belépjen a rendszerbe, hogy módosítsa vagy törölje adatait, illetve, hogy lekérdezze az esetleg elfelejtett jelszót.

Most térjünk át a regisztráló lap tárgyalására!

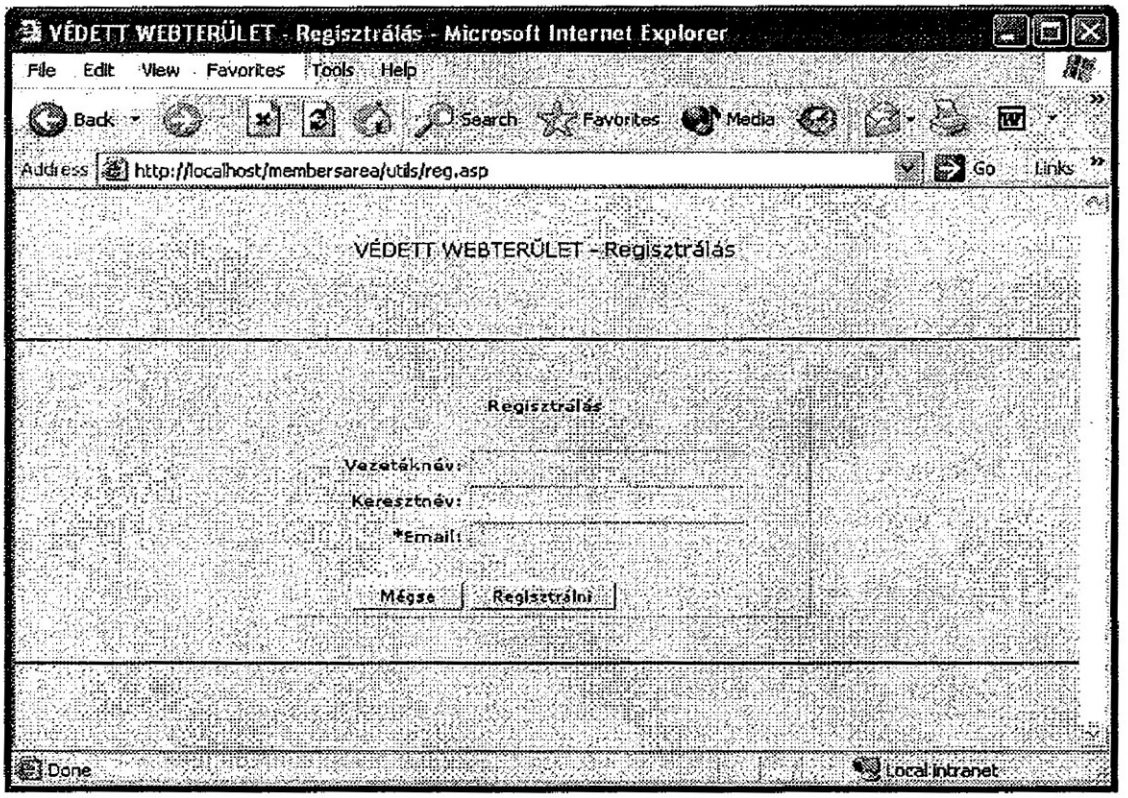
A regisztrálással öt ASP lap foglalkozik: `utils/reg.asp`, `utils/checkreg.asp`, `utils/alreadyreg.asp`, `utils/reguser.asp` és `utils/okreg.asp`.

A `utils/reg.asp` lap magvát egy `<FORM>` HTML elem képezi. Ide három szövegmezőt helyezünk el, kettőt a vezeték-, illetve keresztnév beadására, a harmadikat pedig az email-cím részére. A regisztráláshoz csak az email-cím szükséges. A regisztráló ablakot a 6.2. ábrán láthatjuk.

Ha a felhasználó egy (forma szerint) érvényes email-címet ad be, az alkalmazás a `utils/checkreg.asp` lapban elhelyezett szkripttel legelőször ellenőrzi, hogy létezik-e az adatbázisban egy felhasználó a megadott email-címmel, ugyanis minden email-címre csak egy regisztrálást engedélyezünk. Ha a bejelentkező már regisztrálva van, akkor a `utils/alreadyreg.asp` lappal ezt tudomására hozzuk és megadjuk a jelentőséget, hogy visszamenjen a kezdőlapra. Ha a látogató sikeresen beregisztrálta magát, tudomására hozzuk a `utils/okreg.asp` lappal.

A beadott adatok ellenőrzését és a tulajdonképpeni regisztrálást a `utils/checkreg.asp`, illetve a `utils/reguser.asp` lap végzi el. Ezért a `utils/reg.asp` lapban a `<FORM>` elem `ACTION` attribútuma a `utils/checkreg.asp` lapra utal. A `<FORM>` `METHOD` attribútumával meghatározzuk a beadott adatok átirányításának módszerét. Két lehetőség között választhatunk. Ha `METHOD = GET`, akkor az adatokat a `Request` objektum `QuerySrting` gyűjteményébe írjuk, ha viszont `METHOD = POST`, akkor az adatok a `Request` objektum `Form`

gyűjteményébe kerülnek. A `POST` metódus biztonságosabb, mert a böngészőben nem látszik a lekérdező URL.



6.2. ábra A regisztráló lap

Hibás vagy hiányos adatok beadása esetében a `utils/checkreg.asp` lap visszairányítja a látogatót a regisztráló lapra. Itt, a visszajelzett hiba függvényében írjuk ki a sikertelen próbálkozás okát.

```
<TR VALIGN=baseline>
  <TD ALIGN="left" COLSPAN="2" >
      <%
          Dim msg
          Dim strErrMsg
          Msg = Request.QueryString("err")
          Select Case msg
              Case "01"
                strErrMsg = _
                "Kérem adjon be egy email-címet!"
              Case "02"
                strErrMsg = _
      "Kérem adjon be egy érvényes email-címet! " & _
      "(nev@domain.ext - csak kisbetűkkel)!"
              End Select
        %>
        <SPAN CLASS=head3><FONT COLOR=red>
        <%= strErrMsg %></FONT></SPAN>
    </TD>
</TR>
```

6.3 A beadott adatok ellenőrzése

Az előző szakaszban megismertük alkalmazásunk felépítését és a kezdőlappal, illetve a regisztráló lappal foglalkoztunk. Láttuk, hogy a regisztráló lapon beadott adatok ellenőrzéséről a `utils/checkreg.asp` lap gondoskodik. A tulajdonképpeni regisztrálást a `utils/reguser.asp` lap bonyolítja le. Ebben a szakaszban ezekkel a kérdésekkel foglalkozunk.

A `utils/checkreg.asp` lap szkriptje lényegében csak a beadott email-címet ellenőrzi: beadott-e a felhasználó egy email-címet, illetve "helyes" címet adott-e be? Az első kérdésre a `Request.Form` gyűjtemény `"EmailAddress"` tagja adja meg a választ. A második kérdés magyarázatra szorul. "Jó" vagy "helyes" címnek azokat a címeket tekintjük, melyek formája a következő: `name@domain.ext`. Pontosabban fogalmazva, ez a következőket jelenti

- a címet az `@` karakter pontosan két részre osztja;
- egy rész sem lehet üres;
- mindegyik rész csak az

  `"abcdefghijklmnopqrstuvwxyz_-.0123456789"`

  karakterláncba foglalt karakterekből állhat;
- a `.` (pont) karakter nem lehet egy rész első vagy utolsó karaktere;
- a második részben kell egy `.` (pont) karakternek lenni;
- a címben egy `.` (pont) után nem következhet még egy `.` (pont);
- a második rész `ext` része csak kettő, három vagy négy karakterből állhat.

Ha mindezt figyelembe vesszük, akkor könnyen kódba foglalhatunk egy `IsEmailValid(strEmail)` függvényt, mely ezeket a kikötéseket ellenőrzi. A függvény egy paramétert vár, az ellenőrizendő email-címet. Ha a cím "jó", visszatérési értéke **`True`**, különben **`False`**. Érdemes ezt a függvényt egy külön ASP állományba írni, hogy esetleg más alkalmazásban könnyen felhasználhassuk. Az `IsEmailValid(strEmail)` függvényt a `utils/verifyemail.asp` állományba helyeztem el. Forráskódját itt láthatjuk:

```
<%
' ellenőrzi az email-cím helyességét;
' helyes cím esetén True értéket ad vissza,
' különben False értéket ad vissza
Function IsEmailValid(strEmail)

  Dim strArray
  Dim strItem
  Dim i
```

```
  Dim c
  Dim blnIsValid

  ' abból indulunk ki, hogy az cím helyes
  blnIsValid = True

  ' felbontjuk részeire: name@domain.ext,
  ' azaz pontosan két része kell legyen
  strArray = Split(strEmail, "@")

  ' több vagy kevesebb, mint két része van
  If UBound(strArray) <> 1 Then
        blnIsValid = False
        IsEmailValid = blnIsValid
        Exit Function
  End If

  ' megvizsgáljuk rendre a két részt
  For Each strItem In strArray
        ' egy rész sem lehet üres
        If Len(strItem) <= 0 Then
              blnIsValid = False
              IsEmailValid = blnIsValid
              Exit Function
        End If

        ' sorra vesszük a rész karaktereit
        ' minden részben csak
        "abcdefghijklmnopqrstuvwxyz_-."
        ' vagy számjegyek lehetnek
        For i = 1 To Len(strItem)
              ' csak kisbetűket engedélyezünk
              c = Mid(strItem, i, 1)
              ' "illegális" karakter van a címben
              If _
InStr("abcdefghijklmnopqrstuvwxyz_-.", c) <= 0 _
And Not IsNumeric(c) Then
                    blnIsValid = False
                    IsEmailValid = blnIsValid
                    Exit Function
              End If
        Next

        ' a rész első vagy utolsó karaktere
        '  nem lehet . (pont)
        If Left(strItem, 1) = "." Or _
              Right(strItem, 1) = "." Then
              blnIsValid = False
              IsEmailValid = blnIsValid
              Exit Function
        End If
  Next
```

```
    ' a második részben (domain.ext) kell
    ' egy . (pont) legyen
    If InStr(strArray(1), ".") <= 0 Then
          blnIsValid = False
          IsEmailValid = blnIsValid
          Exit Function
    End If

    ' kiszámítjuk az ext hosszát
    i = Len(strArray(1)) - InStrRev(strArray(1), ".")
    ' az ext hossza csak 2, 3, vagy 4 lehet
    ' (figyelembe véve az új "info" ext-et is)
    If i <> 2 And i <> 3 And i <> 4 Then
          blnIsValid = False
          IsEmailValid = blnIsValid
          Exit Function
    End If

    ' a címben a . (pont) után nem következhet . (pont)
    If InStr(strEmail, "..") > 0 Then
          blnIsValid = False
          IsEmailValid = blnIsValid
          Exit Function
    End If

    ' minden, jó ha a vége jó
    IsEmailValid = blnIsValid

    End Function
%>
```

Ezt az állományt, illetve a benne foglalt függvényt, a `utils/checkreg.asp` lap felhasználja, ezért mindjárt az állomány elején beillesztjük.

Ha a regisztrálásnál hibás vagy hiányos adatokat adott be a látogató, akkor a `Response.Redirect` metódussal visszairányítjuk a regisztráló lapra. A `Redirect` metódus alkalmas arra, hogy a paraméterként megadott céllap URL-jéhez egy lekérdezési láncot (*query string*) hozzákapcsoljunk. Ezt a `Server.Transfer` metódussal nem tehetjük meg. A lekérdezési láncot *név=érték* párokként, a párokat `&` karakterrel elválasztva, rakjuk össze és `?` karakterrel kötjük a céllap URL-jéhez:

```
Response.Redirect _
"maslap.asp?name1=value1&name2=value2"
```

Esetünkben csak egy *név=érték* párt továbbítunk: az `err` nevet egy egész számmal párosítjuk, mely a bekövetkezett hibát azonosítja. Az előző szakaszban láttuk, hogy a hibakód alapján megfelelő hibajelentéssel értesítjük a látogatót, milyen hibát követett el a regisztrálásnál.

Ha a beadott adatok minden szempontból megfelelnek, akkor regisztrálhatjuk új látogatónkat.

6.4 Új tag regisztrálása

A beadott adatokat a `utils/checkreg.asp` lap munkamenetszintű változókba mentette:

```
' tároljuk az adatokat
strFirstName = Request.Form("FirstName")
Session("strFirstName") = strFirstName
strSecondName = Request.Form("SecondName")
Session("strSecondName") = strSecondName
strEmailAddress = Request.Form("EmailAddress")
Session("strEmailAddress") = strEmailAddress
```

Ezek a változók az egész munkamenet alatt, ameddig nem töröljük őket, az alkalmazás minden lapjáról elérhetők. Tehát a regisztrálást végrehajtó `utils/reguser.asp` lapon belül is elérhetők. Ahhoz, hogy a regisztrálást végrehajtsuk, még két dologra van szükségünk. Legelőször is meg kell bizonyosodnunk arról, hogy a látogató nincs-e már regisztrálva. Azonkívül gyártanunk kell egy jelszót a látogató részére. Ha a látogató már regisztrálva van, akkor a `utils/alreadyreg.asp` lapra irányítjuk és közöljük vele ezt a tényt. Ellenkező esetben előkészíthetjük a regisztrálást egy jelszó előállításával.

A jelszót egy eseteleges vagy véletlenszerű (*random*) számmal adjuk meg. Azért, hogy ez a jelszó kevésbé látsszon számnak, a szám hexadecimális alakját fogjuk használni. Ha azt akarjuk, hogy ez az első jelszó hét vagy nyolc karakterből álljon, akkor elegendő egy olyan számintervallumból kiindulni, melynek a kezdőpontja egy hét számjegyű hexadecimális szám és végpontja egy nyolc számjegyű hexadecimális szám. Ügyelnünk kell azonban arra, hogy az intervallum elég nagy legyen. Én a 100.000.000 (&H5F5E100) és 1.000.000.000 (&H3B9ACA00) számokat használtam, de természetesen más intervallum is elképzelhető. Az esetleges számot a VBScript `Random` utasításával és `Rnd` függvényével állítjuk elő. A `Random` utasítás elindítja a véletlenszerű szám generálását, a `Rnd` függvény pedig a következő képlettel, az *`alsóhatár`* és *`felsőhatár`* által meghatározott számintervallumban, egy véletlenszerű számot generál:

```
Int(felsőhatár - alsóhatár + 1) * Rnd + alsóhatár)
```

Ha kísérletezni kívánunk ezzel a függvénnyel, használhatjuk az alábbi `random.asp` egyszerű szkriptet.

```
<!----------random.asp------------------------------>
<TABLE border=0>
<TR>
<TD> </TD><TD><b>decimális</b></TD><TD>
<b>hexadecimális</b></TD>
```

```
</TR>
<TR>
<TD><b>fels&#245;határ</b></TD><TD><%=&H3B9ACA00%></TD><TD>
<%=Hex(&H3B9ACA00)%></TD>
</TR>
<TR>
<TD><b>alsóhatár</b></TD><TD><%=&H5F5E100%></TD><TD>
<%=Hex(&H5F5E100)%></TD>
</TR>
<%
Randomize
r = _
Int((&H3B9ACA00 - &H5F5E100 + 1) * Rnd + &H5F5E100)
%>
<TR>
<TD><b>eredmény</b></TD><TD><%=r%> </TD><TD><%=Hex(r)%></TD>
</TR>
</TABLE>
```

Ha a letöltött ASP lapot ismételten frissítjük, újabb és újabb véletlenszerű számot kapunk.

A véletlenszerűen generált jelszót egy munkamenetszintű változóba mentjük:

```
' gyártunk egy jelszót
Randomize
strPassword = _
Hex((&H3B9ACA00 - &H5F5E100 + 1) * Rnd + &H5F5E100)
' át kell vigyük egy másik lapra,
' ezért egy munkamenetszintű változóba mentjük
Session("strPassword") = strPassword
```

Ennek az eljárásnak az indoklásáról később, a következő szakaszban, még szó esik.

Most vizsgáljuk meg magát a regisztrálást!

A fejezet elején megismert `membersdb.mdb` adatbázis egyetlen egy `members` tábláját kétszer kérdezzük le: egyszer egy választó lekérdezéssel megállapítjuk, hogy a megadott email már szerepel-e a táblában, másodszor pedig, ha az első kérdésre nemleges a válasz, egy hozzáfűző lekérdezéssel a táblához fűzzük a látogató adatait és a számára gyártott jelszót. Mivelhogy az alkalmazás folyamán többször kell majd hasonló adatbázis-műveleteket elvégeznünk, célszerű a szükséges lépéseket általánosítani és külön eljárásokba vagy függvényekbe összefogni. Az így létrejött eljárásokat és függvényeket egy külön ASP állományba gyűjtünk, hogy bármikor beilleszthessük, ha szükség van rá. Jelen esetben a `utils/db.asp` állományról van szó, amit mindjárt az elején a `utils/reguser.asp` állományba beillesztettünk.

Az adatok kezelésére az ADO (ActiveX Data Object) objektummodellt fogjuk használni. A továbbiakban az ADO objektummodell legalább vázlatos ismeretét feltételezem.

A tennivalókat a `ConnectionDB` eljárásba és a `GetRS` függvénybe gyűjtöttem össze:

```
<!--------utils/db.asp--------------->
<!--------#include file=adovbs.inc -->
<%

Dim objConnection
Dim objRecordSet

Set objConnection = _
Server.CreateObject("ADODB.Connection")
Set objRecordSet = _
Server.CreateObject("ADODB.Recordset")

Sub ConnectDB()
  Dim dbPath
  dbPath = "../dbase/membersdb.mdb"

  On Error Resume Next
  ' kapcsoljuk az adatbázist
  objConnection.Open _
   "PROVIDER=MICROSOFT.JET.OLEDB.4.0;DATA SOURCE=" & _
   Server.MapPath(dbpath)

  If Err Then
    Response.Write( _
   "<CENTER><h1><FONT COLOR=red>HIBA TÖRTÉNT: </FONT></h1>" & _
   "<H2>Nem tudtam kapcsolni az adatbázist!</H2></CENTER>" & _
   "<H2>Hibaleírás: </H2><H2><i>" & Err.Description & "</i></H2>")
        Err.Clear
        Response.End
  End If

End Sub

Function GetRS(strSQL)

  On Error Resume Next

  ' lekérdezünk egy rekordhalmazt
  objRecordSet.Open strSQL, objConnection, adOpenStatic, _
adLockOptimistic

  If Err Then
        Response.Write(" _
        <CENTER><h1><FONT COLOR=red>HIBA TÖRTÉNT: </FONT></h1>" & _
```

```
            "<H2>SQL: " &  strSQL & "</H2></CENTER>" & _
            "<H2>Hibaleírás: </H2><H2><i>" & _
            Err.Description & "</i></H2>")
            Err.Clear
            Response.End
    End If

    GetRS = objRecordSet

End Function

%>
```

Az `utils/db.asp` állományban levő szkriptet azzal kezdjük, hogy létrehozzuk az `objConnection Connection` típusú objektumot és az `objRecordSet Recordset` típusú objektumot. A `ConnectionDB` eljárás megteremti a kapcsolatot az adatbázissal. Az eljárásban definiált `dbPath` változóba tároljuk az adatbázis (relatív) virtuális elérési útját. Az `objConnection` objektum `Open` metódusának paramétereként meg kell adjuk a kapcsolatteremtéshez szükséges adatokat. Elegendő, ha a szolgáltatót (*provider*) és az adatforrást (*data source*) adjuk meg, a többit az alapértelmezett értékekre bízzuk. Az adatforrás az adatbázis fizikális elérési útja. Fejlesztési időben ezt viszont nem ismerjük, nem tudjuk, hol helyezkedik majd el az alkalmazásunk a webszolgáltatónk gépén. Ez az oka annak, amiért nem használhatjuk az adatbázis gépünkön érvényes fizikális elérési útját. A `Server` objektum `MapPath` metódusa segítségünkre jön. A `MapPath` metódus ugyanis átalakít egy virtuális elérési utat a megfelelő fizikális elérési útra. Egy reális esetben a `dbase` mappa, amelyikbe adatbázisunkat elhelyezzük, nem szabad közvetlenül elérhető legyen a látogatók számára. A webszolgáltatók minden kiszolgált webhely részére biztosítanak ilyen mappákat. Példaalkalmazásunk esetében nem használtam ilyen privát mappát mert a forráskód úgy is mindenki által megtekinthető. Ha figyelmesen megnézzük a kódot, felvetődik egy kérdés. Miért nem definiáltam a `dbPath` változót a többi "globális" adattal együtt az `utils/db.asp` állományban? Erre a kérdésre is a feleletet később, a következő szakaszban fogjuk megkapni.

Most nézzük meg a `GetRS` függvényt! A függvény paramétere a `strSQL` lekérdezési karakterláncot adja át a függvénynek. Ez a lekérdezési karakterlánc a `members` táblára vonatkozó választási, hozzáfűző, frissítő vagy törlő lekérdezés lehet. A függvény megnyitja a létrehozott `objRecordSet` rekordhalmazt, azaz elvégzi a lekérdezést. Ha választási lekérdezésről volt szó, a visszaadott `objRecordSet` rekordhalmaz gyűjteményei az eredményhalmazzal vannak megtöltve. A `Recorset` objektum `Open` metódusának több alakja használható. Ebben az esetben a paraméterekkel megadjuk a lekérdezési karakterláncot (*query string*), a kapcsolatot (*connexion*), a kurzor típusát, valamint a lezárás típusát. A kurzort az `adOpenStatic` (statikus kurzor), a lezárást az `adLockOptimistic` (optimista lezárás: egyszerre több felhasználó is módosíthatja a

táblát) konstansok határozzák meg. Ezek a konstansok a `utils/adovb.inc` állományban vannak megadva. Ha megnézzük az állományt, láthatjuk, hogy Microsoft készítette és az ADO-konstansokat tartalmazza VBScript-ben. (Természetesen létezik egy `adojs.inc` állomány is az ADO-konstansokkal JScript-ben.)

Miután ezeket a dolgokat tisztáztuk, visszatérhetünk a `utils/reguser.asp` lap tárgyalására. Azzal kezdjük, hogy kapcsolatot hozzunk létre az adatbázissal, majd egy választó lekérdezéssel megnézzük hány rekordunk van, mely a megadott email-címet tartalmazza:

```
' kapcsoljuk az adatbázist
ConnectDB
Dim strSql
Dim simpleQuote
simpleQuote=chr(39)
' választó lekérdezés
strSql= "SELECT COUNT(*) as Numb " _
     FROM members WHERE Email=" & _
     simpleQuote & Session("strEmailAddress") & _
     simpleQuote
GetRS strSql
Dim intRecNumb
intRecNumb = objRecordSet.Fields("Numb")
objRecordSet.Close
```

Ha a látogató még nincs regisztrálva, gyártunk egy jelszót, adatait hozzáfűzzük a `members` táblához és bezárjuk a kapcsolatot az adatbázissal:

```
Dim strPassword
' másként nem is lehet
If intRecNumb = 0 Then
     ' gyártunk egy jelszót
     Randomize
     strPassword = _
      Hex((&H3B9ACA00 - &H5F5E100 + 1) * Rnd + &H5F5E100)
     ' át kell vigyük egy másik lapra,
     ' ezért egy munkamenetszintű változóba mentjük
     Session("strPassword") = strPassword

     Dim strInsert
     Dim dateDelimiter
     dateDelimiter = "#"
     SetLocale(&H0409) 'US amerikai LCID
     ' hozzáfűző lekérdezés
     strInsert = "INSERT INTO members ( " & _
     "FirstName, SecondName, " & _
     "Email, Password, RegisteredOn ) " & _
      "VALUES (" & _
      simpleQuote & Session("strFirstName") & _
```

```
        simpleQuote & ", " & _
        simpleQuote & Session("strSecondName") & _
        simpleQuote & ", " & _
        simpleQuote & Session("strEmailAddress") & _
        simpleQuote & ", " & _
        simpleQuote & strPassword & simpleQuote & _
        ", " & _
        dateDelimiter & FormatDateTime(Now,2) & _
        dateDelimiter )"
      SetLocale(0) 'alapértelmezett LCID

If Session("strEmailAddress") <> "" Then
    GetRS strInsert
Else
    Set objRecordSet = Nothing
    objConnection.Close
    Set objConnection = Nothing
    Response.End
End If
' rendet csinálunk
Set objRecordSet = Nothing
objConnection.Close
Set objConnection = Nothing
```

A lekérdező karakterlánc felépítésénél ügyelnünk kell az értékek megadásánál. Ha szöveges értékről van szó, akkor idézőjeleket (') kell használnunk, ha dátumról van szó, akkor elválasztójelként a # karaktert kell használnunk. Megszoktam, hogy ezeket a karaktereket változókba mentem, mert így könnyebb a kódot elolvasni:

```
Dim simpleQuote, dateDelimiter
simpleQuote = chr(39)
dateDelimiter = chr(35)
```

A hozzáfűző lekérdezés esetén az aktuális dátumot is beírjuk az adatbázis `RegisteredOn` adatmezőjébe. Az Access adatbázis esetében a dátumot az amerikai írásmód szerint kell beadnunk: mm/dd/yyyy. Ezt a legegyszerűbben úgy érhetjük el, hogy a lekérdezési karakterlánc összeállítása előtt a `SetLocale(&H0409)` utasítással a lokális beállítást átállítjuk az amerikaira, majd a lekérdezési karakterlánc összeállítása után a `SetLocale(0)` utasítással visszaállítjuk az alapértelmezett értékére. Itt is ugyanígy jártam el.

Miután az új tagot regisztráltuk, erről őt is értesítenünk kell. Ezen kívül a jelszót is el kell juttatnunk hozzá! A jelszót egy email segítségével juttatjuk el az új tagnak. Ezzel elkerüljük, hogy valaki idegen vagy nem létező címmel regisztrálja magát. Ha a regisztráló saját email-címét használta, megkapja tőlünk az első jelszót, amit később megváltoztathat. Miután elküldtük a regisztrálást igazoló email-t, a

`utils/okreg.asp` lapon elmondjuk a jelentkezőnek, milyen adatokat tároltunk és hogy az első jelszót egy üzenettel elküldtük.

Az email megszerkesztésével és elküldésével a következő szakaszban foglalkozunk.

6.5 Az email megszerkesztése és elküldése

A második szakaszban láttuk, miként történik a regisztrálás. Láttuk, hogy miután egy új jelentkező megadta adatait, készítünk részére egy jelszót és regisztráljuk az adatbázisban. Az új tagnak a belépésre szüksége van az első, automatikusan létrehozott jelszóra. Ezt a jelszót egy email üzenettel küldjük el címére. Az email elküldésére a CDONTS komponenst fogjuk használni. Ha ez a komponens nem található a gépünkön, akkor a CDOSYS komponenst kell használnunk.

A `user/reguser.asp` szkriptben regisztrálás után a `SendEmail` függvény meghívásával egy email-t küldünk az új tagnak és közöljük vele a sikeres regisztrálást, valamint az első jelszót. A `SendEmail` függvényt a `user/sendemail.asp` állományban definiáljuk:

```
<!-------utils/sendemail.asp------------>
<%
Function SendEmail(strFrom,strTo,strSubject,strBody)
    Dim objNewMail
    On Error Resume Next
    ' létrehozunk egy CDONTS.NewMail objektumot
    Set objNewMail = _
    Server.CreateObject("CDONTS.NewMail")
    If Err.Number<>0 Then
      Response.Write "<HTML>"
      Response.Write "<TITLE> EMAIL FAILURE </TITLE>"
      Response.Write _
      "<meta http-equiv=""Content-Language"" " & _
      "content=""hu"">"
      Response.Write _
      "<meta http-equiv=""Content-TYPE"" " & _
      "content=""text/html; CHARSET=iso-8859-2"" >"
      Response.Write _
      "<!-- #include file=""cssrules.asp"" -->"
      Response.Write "</HEAD>"
      Response.Write "<BODY  text=""#993300"">"
      Response.Write "Error: " & Err.description
      Response.Write _
      "<CENTER><H2><FONT COLOR=red>" &_
      "Az Email elküldésénél a szerveren egy " & _
      "hiba keletkezett!" & "</FONT></H2><BR>"
      Response.Write "<H2>Kérem jelentse ezt " & _
      "egy üzenettel " & _
```

```
            "<a href=mailto:" & strFrom & _
            "?subject=EMAILFAILURE>" & _
            " ezen </a> a címen és írja meg " & _
            "a használt Email címét.</H2>"
            Response.Write "</BODY></HTML>"
            Err.Clear
            Response.End
          End If
          ' kitöltjük a tulajdonságokat
          objNewMail.From = strFrom
          objNewMail.To = strTo
          objNewMail.Subject = strSubject
          objNewMail.Body = strBody

          ' elküldjük az üzenetet
          objNewMail.Send

          If Err.Number<>0 Then
            SendEmail=False
          Else
            SendEmail=True
          End If

          Set objNewMail = Nothing

        End Function
        %>
```

Ha a `CTONTS.NewMail` objektum létrehozásánál egy hiba történt, akkor megszakítjuk a függvény végrehajtását és értesítjük a felhasználót az esetről egy dinamikusan létrehozott lappal. Itt megkérjük a felhasználót, hogy tudomásunkra hozza a hibát, mert ebben az esetben intézkednünk kell.

Ugyanebben az állományban találjuk a `SendCDOSYSEmail` függvényt, amit a CDOSYS komponens esetében kell használnunk.

A `utils/sendemail.asp` állományt a `reguser.asp` állományba illesztettük még az állomány elején, tehát a `SendEmail-t` meghívhatjuk ebből az állományból. Az elküldött email egy egyszerű szöveges üzenet. Az üzenet szövegét a `strBody` változó a `body_reguser` változótól kapja és továbbítja a `SendEmail` függvénynek. A `body_reguser` változót viszont a `data.asp` állományban látjuk el értékkel. Láthatjuk, hogy az üzenetszövegnek van egy változatlan és egy változó része. Más szóval az üzenetet dinamikusan szerkesztjük, és munkamenetszintű változókat használunk erre a célra. Ez az oka annak, hogy a jelszót és a jelentkező adatait munkamenetszintű változókba mentettük. Szintén ezzel magyarázható, miért nem lehetett a `utils/data.asp` állományt a `utils/reguser.asp` lap elején a lapba illeszteni.

Ha az üzenetet sikeresen elküldtük, akkor a `SendEmail` függvénynek visszatérési értéke **`True`**, különben **`False`**. A `SendEmail` függvénynek a visszatérési értékét a `utils/reguser.asp` lapon ellenőrizzük:

```
If SendEmail(strFrom,strTo,strSubject,strBody) Then
    Response.Redirect "okreg.asp"
Else
    Response.Redirect "emailfailure.asp?err=reg"
End If
```

Ha sikerült az email-t az automatikusan generált jelszóval elküldeni, átirányítjuk a látogatót a `ustils/okreg.asp` lapra és értesítjük a sikeres regisztrálásról és arról, hogy email-be megkapja az első jelszót. Ha az email elküldésénél valami nem sikerült, akkor a `utils/eamilfailure.asp` lapra irányítjuk át a látogatót. Az átirányításra a `Response.Redirect` metódust használjuk, így egy lekérdezési karakterláncot is átküldhetünk a céllapra. Jelen esetben az `err=reg` név/érték párt küldjük tovább, amit a céllapon kiértékelünk:

```
<%
    Session.Abandon
    Dim msg
    Dim strHref
    Dim strSubject
    msg = Request.QueryString("err")
    Select Case msg
      Case "reg"
        strHref = "reg.asp"
        strSubject = "HIBA A BEJELENTKEZÉSNÉL"
      Case "mod"
        strHref = "mod.asp"
        strSubject = "HIBA AZ ADATMÓDOSÍTÁSNÁL"
      Case "pwd"
        strHref = "forgotpwd.asp"
        strSubject = "HIBA A JELSZÓ ELKÜLDÉSÉNÉL"
    End Select
%>
Próbálja meg
<A HREF="<%= strHref%>">még egyszer</A>.
Ha ezúttal is hiba történik, kérjük értesítsen minket ezen a
<A HREF=mailto:
<%= editors_email%>?subject=<%= strSubject%>> címen.</A>
```

A `utils/sendemail.asp` állományba definiált `SendEmail` függvényt a regisztráláson kívül az adatok módosítása, illetve az elfelejtett jelszó elküldésekor is használni fogjuk. Az esetleg ezekben az esetekben is előforduló hibákról szintén értesítenünk kell a látogatót, így magától kínálkozik a lehetőség ugyanazt a `utils/eamilfailure.asp` lapot használni. Valahányszor a látogatót átirányítjuk erre a lapra, a lap URL-jéhez fűzött lekérdezési karakterlánccal megjelöljük a helyet,

ahonnan az átirányítás történik. A regisztrálásnál az `err=reg`, az adatmódosításkor az `err=mod`, az elfelejtett jelszó megküldésénél pedig az `err=pwd` név/érték párt használjuk. A `utils/eamilfailure.asp` lap tehát pontosan "tudja" melyik műveletnél történt a hiba és felajánlhatja a látogatónak, hogy próbálja meg még egyszer a műveletet végrehajtani.

A regisztráció ezzel befejeződött. A következő szakaszban a bejelentkezésről, az adatok módosításáról és törléséről lesz szó.

6.6 Az adatok módosítása és törlése, a jelszó lekérdezése

Ebben a szakaszban az adatok módosításával, a bejelentkezéssel és az adatok törlésével fogunk foglalkozni. Alapjában véve új dolgokat nem is kell megismernünk: minden, amit a regisztrációval kapcsolatban láttunk, itt kevés módosítással alkalmazható.

Egy új látogató regisztrálta magát, adatait felvettük az adatbázisba, elküldtünk neki egy email-t az automatikusan generált első jelszóval és értesítettük a sikeres regisztrációról. A látogató most a megkapott jelszót meg akarja változtatni, esetleg az regisztrálásnál beadott adatain is módosításokat szeretne eszközölni. Nézzük meg miként történik az adatok módosítása!

A módosítással a következő állományok foglakoznak: `utils/mod.asp`, `utils/checkmod.asp`, `utils/showdata.asp`, `utils/checkdata.asp`, `utils/moduser.asp` és `utils/okmod.asp`. A `utils/mod.asp` lapon a látogatónak be kell adnia az email-címet és a jelszót, amit a jelen esetben a regisztrációt igazoló üzenettel kapott tőlünk. Amint látjuk, egy egyszerű lapról van szó, egy `<FORM>` elemmel, mely a beadott adatokat `POST` metódussal a `utils/checkmod.asp` állománynak továbbítja ellenőrzésre. A `utils/checkmod.asp` állomány ellenőrzi a beadott adatokat. Ha a látogató nem adott be email-címet vagy jelszót, vagy rossz email-címet adott be, akkor a `utils/mod.asp` lapra irányítja a látogatót. Ellenkező esetben lekérdezi az adatbázist: hány bejegyzés létezik a beadott email-címmel és jelszóval.

```
' kapcsoljuk az adatbázist
ConnectDB
' keressük a beadott adatokat:
' hányszor fordulnak elő?
Dim strSql
Dim simpleQuote
simpleQuote=chr(39)
strSql= "SELECT COUNT(*) as Numb FROM " & _
        "members WHERE Email=" & _
        simpleQuote & strEmailAddress & _
        simpleQuote & " AND Password=" & _
```

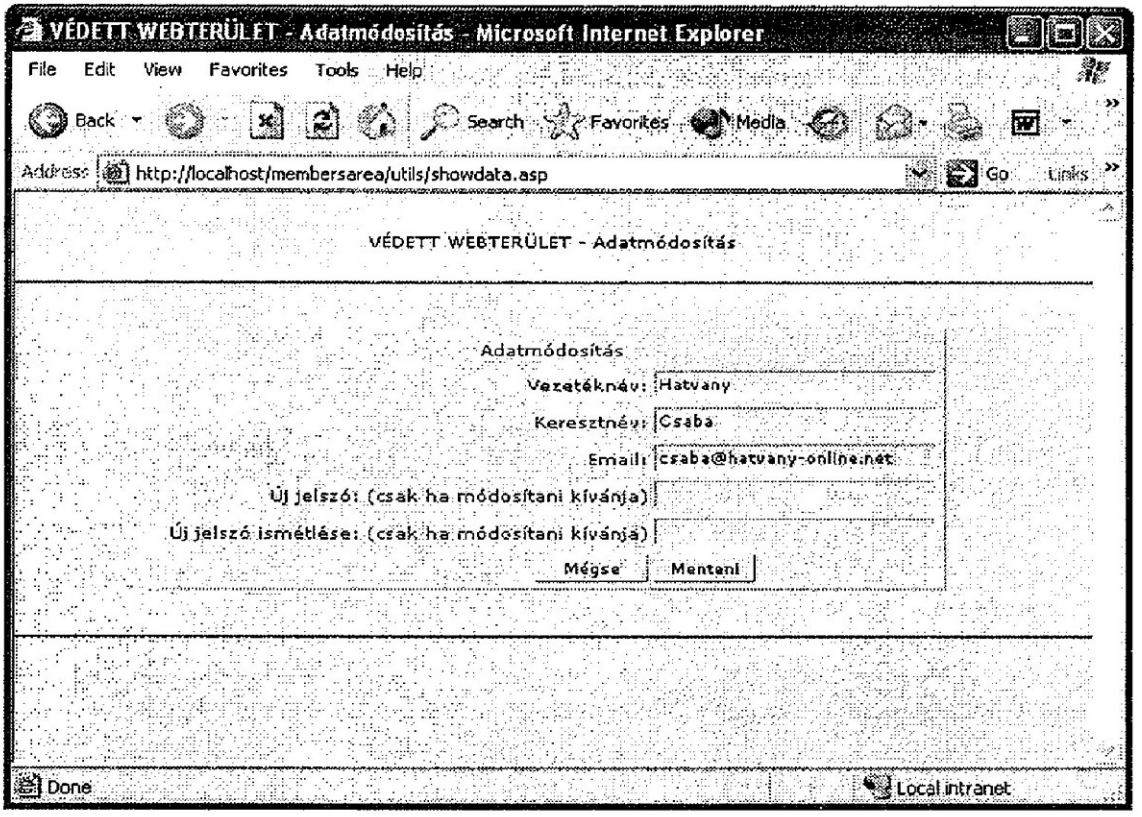
```
            simpleQuote & _
            strPassword & simpleQuote

    GetRS strSql

    Dim intRecNumb
    intRecNumb=objRecordSet.Fields("Numb")
    objRecordSet.Close
```

Elméletileg itt három lehetőség adódik.

Kezdjük el a legrosszabbal: egynél több rekord van az adatbázisban, ami a beadott adatoknak megfelel. Ennek az esetnek nem szabad előfordulnia! Ha mégis bekövetkezik, értesítjük a látogatót és elhagyjuk a `utils/checkmod.asp` lapot. Ebben az esetben a rendszergazdának intézkednie kell. Ha az adatbázisban nem találunk egy rekordot sem, mely a beadott adatoknak megfelelne, visszairányítjuk a látogatót a `utils/mod.asp` lapra. Végül, ha a látogató regisztrálva van, azaz az előbbi lekérdezés pontosan egy rekordot eredményezett, akkor egy további lekérdezéssel kiolvassuk az adatait az adatbázisból és munkamenetszintű változókba mentjük:

```
    ' munkamenetszintű változóba mentjük
    Session("strUserFirstNameOrEmail") = _
            strUserFirstNameOrEmail
    Session("strFirstName") = _
            objRecordSet.Fields("FirstName")
    Session("strSecondName") = _
            objRecordSet.Fields("SecondName")
    Session("strEmailAddress") = _
            objRecordSet.Fields("Email")
    Session("strPassword") = _
            objRecordSet.Fields("Password")
    Session("strUID") = _
            objRecordSet.Fields("UID")
```

Ezek után a `utils/showdata.asp` lap segítségével megmutatjuk az adatokat és megadjuk a lehetőséget, hogy ezeket megváltoztassa. A `utils/showdata.asp` lapot a 6.3. ábrán láthatjuk.



6.3. ábra Az adatok módosítása

A `utils/showdata.asp` lap felépítése szintén egy `<FORM>` elemre alapszik, mely `POST` metódussal átjuttatja a módosított adatokat a `utils/checkmod.asp` lapra ellenőrzés végett. Ez a `<FORM>` megmutatja a látogató vezetéknevét, keresztnevét és email-címét, úgy ahogy ezek az adatbázisban szerepelnek. A vezetéknevet és a keresztnevet a regisztrált tag módosíthatja, esetleg ki is törölheti. Az email-címet viszont nem szabad, hogy megváltoztassa, ezért a megfelelő szövegmezőbe nem engedjük meg, hogy írjon. Ezt a `READONLY` attribútummal érjük el:

```
<INPUT NAME="EmailAddress" TYPE="text"
VALUE="<%=Session("strEmailAddress") %>"
SIZE=30 READONLY>
```

Az említett szövegmezőkön kívül, két további szövegmező gondoskodik új jelszó beadásáról. Miután a kívánt módosítást ellenőriztük és mindent rendben találtunk, elvégezhetjük a felhasználó által kért módosításokat. A módosításokat a `utils/moduser.asp` lap hajtja végre. Attól függően, hogy a felhasználó megváltoztatja-e jelszót vagy nem, két különböző frissítő lekérdezést használunk:

```
<%
' kapcsoljuk az adatbázist
ConnectDB

Dim strUpdate
Dim simpleQuote
```

```
impleQuote=chr(39)
If Session("strPassword")="" Then
    strUpdate="UPDATE members SET FirstName = " & _
    simpleQuote & _
    Session("strFirstName") & simpleQuote & "," & _
    "members.SecondName = " & simpleQuote & _
    Session("strSecondName") & simpleQuote & "," & _
    "members.Email = " & simpleQuote & _
    Session("strEmailAddress") & _
    simpleQuote & " " & _
    "WHERE (members.UID)=" & _
    Session("strUID")
Else
    strUpdate="UPDATE members SET FirstName = " & simpleQuote & _
    Session("strFirstName") & simpleQuote & "," & _
    "members.SecondName = " & simpleQuote & _
    Session("strSecondName") & simpleQuote & "," & _
    "members.Email = " & simpleQuote & _
    Session("strEmailAddress") & simpleQuote & "," & _
    "members.[Password] = " & simpleQuote & _
    Session("strPassword") & simpleQuote & " " & _
    "WHERE (members.UID)=" & Session("strUID")
End If
If Session("strUID")<>"" Then
    GetRS strUpdate
Else
    Set objRecordSet = Nothing
    objConnection.Close
    Set objConnection = Nothing
    Response.End
End If
' rendet csinálunk
Set objRecordSet = Nothing
objConnection.Close
Set objConnection = Nothing
Dim strBody
Dim strSubject
%>
<!--    #include file="data.asp"    -->
<%
strSubject = pagetitle_moduser
strBody = body_moduser

Dim strFrom
Dim strTo
strFrom = editors_email
strTo = Session("strEmailAddress")
If SendEmail(strFrom,strTo,strSubject,strBody) Then
    Response.Redirect "okmod.asp"
Else
    Response.Redirect "emailfailure.asp?err=mod"
End If
%>
```

Mint láttuk, miután a utils/checkmod.asp lap ellenőrizte a beadott adatokat és ha rendbe találta, az adatbázisból lekérdezett adatokat munkamenetszintű változókba menti, többek között az UID adatmező tartalmát is:

```
Session("strUID") = objRecordSet.Fields("UID").
```

A utils/moduser.asp lapban csak akkor kérdezzük le az adatbázist, ha a munkamenetszintű Session("strUID") változó nem üres. Miért szükséges ez az ellenőrzés? Lehet-e egyáltalán ez a változó üres?

A munkamenetszintű Session("strUID") változó azután kapja az értékét, miután a látogató a utils/mod.asp lapon beadta az email-címét és a jelszót és megnyomta az **"Adatokat módosítani"** gombot, ezzel letöltve a utils/checkmod.asp lapot, mely a beadott adatokat helyesnek találta. Ekkor jelenik meg a utils/showdata.asp lap, mely "ismeri" már a Session("strUID") változó értékét. Ekkor viszont fenn áll az a lehetőség, hogy látogató egy időre megszakíthatja az alkalmazással való kommunikációt, azaz képernyőjén nyugodni hagyja a utils/showdata.asp lapot a böngészőjében és nem csinál semmit, vagy más alkalmazással foglakozik. Alapértelmezés szerint, ha a látogató és az alkalmazás 20 percig nem kommunikálnak, az aktuális munkamenet véget ér és a munkamenetszintű változók elveszítik értékeiket. Ezt az esetet védjük ki a Session("strUID") változó ellenőrzésével.

Miután a látogató által kívánt módosítás megtörtént, értesítjük a látogatót és elküldünk a címére egy üzenetet, melyben összefoglaljuk a módosított adatokat. Ezt a feladatot a utils/okmod.asp látja el.

A fentiekből az is kiderül, hogy egy regisztrált tag, nem tudja módosítani az egyszer beadott email-címét. A valóságban viszont előfordul, hogy valaki megváltoztatja a címét. Az egyszerűség kedvéért, ezt az esetet egy kis kitérővel oldottam meg: mint látni fogjuk, a felhasználó törölheti adatait az adatbázisból és azután még egyszer regisztrálhatja magát, ezúttal az új címével.

A rendszerbe való belépés, a rendszerből való törlés, illetve a jelszólekérdezés alapjában véve nem tér el a regisztrálástól vagy az adatok módosításától. A belépőlap a utils/login.asp. A beadott adatokat ebben az esetben a utils/checkuser.asp ellenőrzi. Ha minden rendben van, akkor a látogatót beeresztjük a utils/showprotected.asp lapra, ahonnan automatikusan továbbkerül a "védett területre". Ugyanez érvényes az adatok törlésére, illetve az elvesztett vagy elfelejtett jelszó kérésére is. A utils/del.asp lap lekérdezi a regisztrált tag email-címét, a utils/checkdel.asp lap pedig ellenőrzi a beadott adatot és ennek függvényében reagál. A kezdőlapon (default.asp), a belépőlapon (utils/login.asp), az adatokat módosító lapon (utils/mod.asp) és az adatokat törlő lapon

(utils/del.asp) egy-egy hivatkozás (*link*) található az utils/forgotpwd.asp lapra. A utils/forgotpwd.asp kérdőívbe beadott email-címet a utils/checkpwd.asp ellenőrzi és ha megtalálta az adatbankban az email-címnek megfelelő jelszót, elküldi egy üzenettel a megadott címre.

6.7 Az állományok megvédése

Itt álljunk meg egy pillanatra és vizsgáljuk meg közelebbről, hogy történik a védett terület biztosítása? Mielőtt a látogatót átirányítjuk a utils/showprotected.asp lapra, a utils/chekuser.asp lap a Session("UserLoggedIn") munkamenetszintű változó értékét **True** értékre állítja. A utils/showprotected.asp lap első dolga, hogy ennek a munkamenetszintű változónak ez értékét ellenőrizze és ettől függően beeressze a látogatót a védett területre vagy nem. A protected mappába elhelyezett állományok mindegyike hasonlóképpen ellenőrzi, hogy a látogató a "főbejáraton" lépett-e be a rendszerbe vagy meg akarja kerülni ezt. Ez utóbbi esetben visszaküldi a betolakodót a kezdőlapra. Emlékezzünk vissza, hogy a default.asp kezdőlap a Session.Abandon metódussal az aktuális munkamenetet befejezi és ezzel megsemmisíti az összes munkamenetszintű változót.

Felvetődik még egy fontos kérdés: mennyire védettek a utils mappa állományai? A utils mappa állományairól a következőt mondhatjuk el. Egyes állományt, mint a loginadmin.asp, login.asp, mod.asp, forgotpwd.asp, del.asp, alreadyreg.asp, emailfailure.asp, okdel.asp, okmod.asp, okreg.asp és sentpwd.asp, nem kell egyáltalán megvédeni, mivelhogy ezek csupán egy kérdőívet vagy egy egyszerű üzenetet jelenítenek meg. Más állományt, mint a check*.asp állományokat, saját forráskódja védi meg hívatlan vendégtől. Ezek az állományok ellenőrzik a Request.Form gyűjtemény "EmailAddress" tagját és ha üres, akkor visszairányítják a látogatót a megfelelő kérdőívre. Így például a checkmod.asp közvetlen lehívása ugyanazt eredményezi, mint a mod.asp lehívása. Egy megint más kategóriát alkotnak azok az ASP állományok, melyek változókat vagy függvényeket definiálnak, illetve értékekkel látják el a változókat, de nem hívnak meg semmilyen függvényt és nem jelenítenek meg semmit. Ezek közvetlen lehívása egy üres HTML lapot eredményez és különben sem ronthatnak semmit az alkalmazáson. Ebbe a kategóriába tartoznak a cssrules.asp, data.asp, db.asp, sendemail.asp és a verifyemail.asp állományok. Végül, egy legutolsó és igen fontos kategóriát alkotnak azok az ASP állományok, melyek bizonyos műveleteket, akciókat hajtanak végre. Itt ügyelnünk kell arra, hogy hiányos vagy nem megfelelő adatok nehogy kárt okozzanak az adatbázisban. Különös figyelemmel kell tehát a reguser.asp, moduser.asp, mailer.asp, showmember.asp és showprotected.asp állományokat kezelni. A showprotected.asp állományról már beszéltünk. A reguser.asp és moduser.asp állományok esetében láttuk, hogy csak akkor végzik

el a hozzáfűző, illetve frissítő lekérdezést, ha a `Session("strEmailAddress")`, illetve `Session("strUID")` munkamenetszintű változók nem üresek. A másik két állománnyal később még találkozni fogunk.

A következő szakaszban a védett terület sajátos állományaival fogunk foglalkozni. A fejezet elején láttuk, hogy a védett területen a rendszergazda különleges jogokkal rendelkezik. Ezt fogjuk részleteiben megvizsgálni.

6.8 A tagok listája

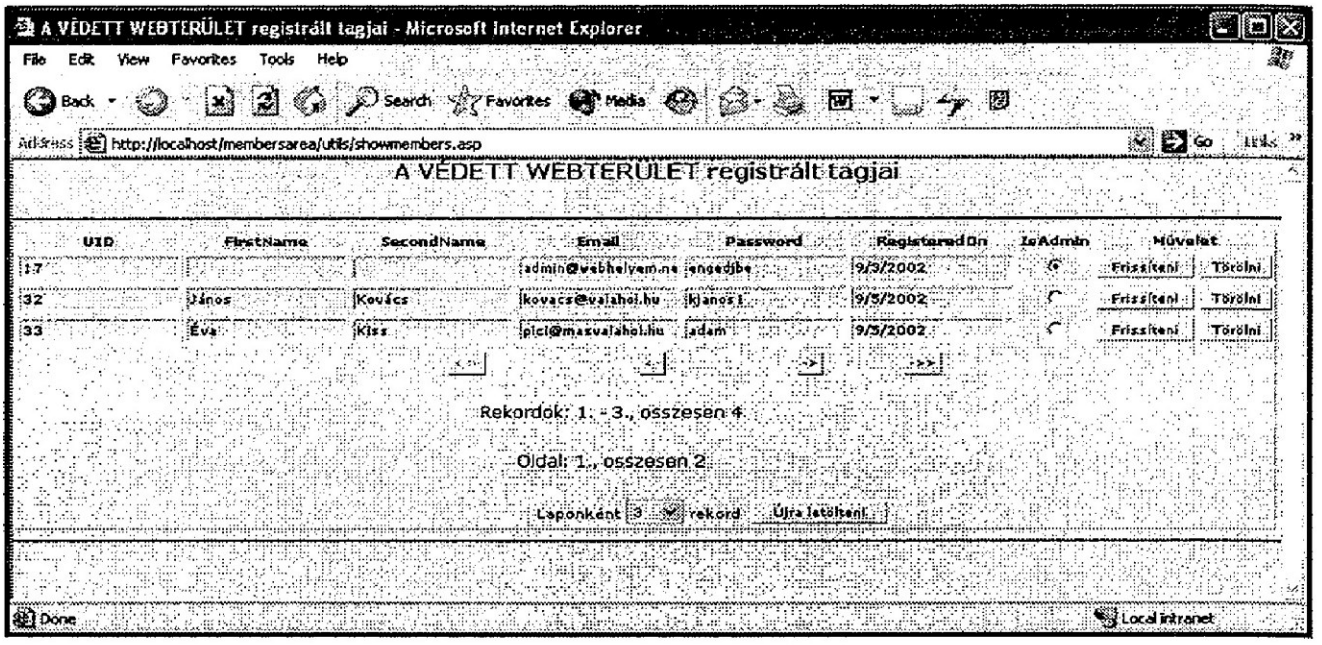
Az előző szakaszokban a rendszerbe való belépést, az adatok módosítását, valamint az adatok törlését tárgyaltuk. Most végre eljutottunk oda, hogy a látogató beléphet a védett területre. Mint tudjuk, a belépőlapon (`utils/login.asp`) beadott adatokat a `utils/checkuser.asp` lap ellenőrzi. Ha a látogató regisztrálva van, akkor a `Session("UserLoggedIn")` munkamenetszintű változónak **`True`** értéket ad és átirányítja a látogatót a védett terület kezdőlapjára (`protected/default.asp`). Ez a lap nagyon egyszerű: felsorolja a védett területen elérhető lapokat és az ezekhez vezető kapcsolatokat (*link*).

```
<!---protected/default.asp----->
<%
  If Not Session("UserLoggedIn") Then
     Response.Redirect "../default.asp"
  End If
%>
<!--     #include file="../utils/data.asp"       -->
<HTML>
<HEAD>
<META NAME="GENERATOR"
Content="Microsoft Visual Studio 6.0">
<TITLE><%= site_title%></TITLE>
<!--     #include file="../utils/cssrules.asp"  -->
</HEAD>
<BODY >
<CENTER>
<BR>
<BR>
<H1><%= site_title%></H1><BR><BR>
<HR>
<BR>
<H2>Üdvözlöm
<%= Session("strUserFirstNameOrEmail")%>
</H2><BR><BR>
</CENTER>
<HR>
```

```
<UL>
  <LI>
    <A href=1stprotectedpage.asp>
    Első védett lap</A>
  </LI>
  <LI>
    <A href=2ndprotectedpage.asp>
    Második védett lap</A>
  </LI>
  <LI>
    <A href=3rdprotectedpage.asp>
    Harmadik védett lap</A>
  </LI>
  <LI>
    <A href=members.asp>Tagok listája</A>
  </LI>
  <LI>
    <A href=newsletters.asp>
    Hírlevelet küldeni</A>
  </LI>
</UL>
<A href="../default.asp">Kilépni</A>
</BODY>
</HTML>
```

Mivelhogy ez egy példaalkalmazás, itt három "fantom" lapot találunk, azaz egyszerű lapokat, csupán egy címmel. Ezen kívül a lap még két kapcsolatot tartalmaz: "Tagok listája" és "Hírlevelet küldeni". Az utóbbi két kapcsolat olyan lapokhoz vezet, melyek megtekintésére csak a rendszergazda jogosult. Jelen részben ezzel a két különlegesen védett lappal foglalkozunk. Mielőtt azonban továbbmennénk, vegyük észre, hogy a védett terület kezdőlapján legelső dolgunk, hogy megbizonyosodjunk: belépett-e a rendszerbe a látogató vagy más úton próbál a laphoz jutni?

A védett terület kezdőlapja a "Tagok listája" kapcsolattal a `protected/members.asp` lapra vezet. Ez a lap is, mint mindegyik a `protected` mappában, először megvizsgálja a `Session("UserLoggedIn")` változó értékét. Amennyiben ez az érték **`True`**, lehívja a `utils/loginadmin.asp` lapot. Ellenkező esetben visszaküldi a hívatlan vendéget az alkalmazás kezdőlapjára. A `utils/loginadmin.asp` lap csak annyiban különbözik a már ismert `utils/login.asp` laptól, hogy mindjárt az elején szintén a `Session("UserLoggedIn")` változó értékét vizsgálja meg. Ezentúl a rendszergazdának az email-címét és a hozzátartozó jelszót várja. A beadott adatok helyességét a `utils/checkadmin.asp` ellenőrzi. A `protected/members.asp` lap a `Session("AdminAction")` munkamenetszintű változóhoz a `"members"` karakterláncot rendelte hozzá. Ha a beadott adatok a rendszergazda adataival megegyeznek, akkor a `utils/checkadmin.asp` lap ennek a változónak alapján megmutatja a rendszergazdának a tagok listáját, azaz a `util/showmembers.asp` lapot (6.4. ábra).



6.4. ábra A tagok listája

A továbbiakban ezt a lapot fogjuk részletesen megtárgyalni. A `util/showmembers.asp` lap egy szép példája a dinamikusan létrehozott ASP lapoknak. Jelentkezzünk be a rendszergazda adataival és nézzük meg a tagok listáját! Nézzük meg a lap forráskódját a letöltött ZIP állományban!

Vizsgáljuk meg először az állomány HTML részét! Itt egy `<FORM>` elem játssza a központi szerepet. Vegyük észre, hogy a `<FORM>` kérdőív `ACTION` tulajdonságával ugyanerre a lapra utal, és a `submit` típusú `<INPUT>` elemek akciójára ugyanezt a lapot a `POST` metódus közbenjárásával tölti ismét le. A `POST` metódus a `<FORM>` elem tagjait továbbítja a letöltendő lapnak. Ezeket a lap megjelenítése előtt kiértékelhetjük és ennek megfelelően alakíthatjuk a megjelenítendő lapot. Mint tudjuk, a `<FORM>` elem tagjait a `Request.Form` gyűjteményben kell keresnünk.

A lap az adatbázis lekérdezése alapján jeleníti meg a `members` tábla tartalmát. A lekérdezés eredményét a `objRecordSet` rekordhalmaz tartalmazza. Egyik célunk az, hogy a felhasználó adhassa meg, hány rekordot szeretne a lapon látni és lapozhasson a rekordhalmazban előre vagy hátra. Erre a célra a `RecordSet` objektum `PageSize` és `PageCount` tulajdonságát használjuk. A `PageSize` meghatározza, illetve visszaadja, hány rekord van egy táblalapon. A `PageCount` tulajdonságot csak olvasni tudjuk. Értéke a táblalapok száma, mely a rekordhalmazban szereplő rekordok száma és a `PageSize` értékéből adódik. Alapértelmezésben a `PageSize` értékét 3-ra állítjuk. Ezt az értéket és az ebből adódó `PageCount` értéket a `intPageSize`, illetve `intPageCount` változókba tároljuk, további használatra.

```
If Request("pagesize")<>"" Then
```

```
    ' rekord/lap
    intPageSize = CInt(Request("pagesize"))
  Else
    ' alapértelmezés 3 rekord/lap
    intPageSize = 3
  End If

  objRecordSet.PageSize = intPageSize
  ' lapok száma
  intPageCount = objRecordSet.PageCount
```

A `RecordSet` objektum `AbsolutePage` tulajdonsága az aktuális táblalapot határozza meg. Ezt az értéket a `intPage` változóval adjuk meg és kezdeti értékét `1`-re tesszük:

```
objRecordSet.AbsolutePage = intPage
```

Most minden készen áll a lap első megjelenítésére.

A tábla megjelenítését a tábla fejlécével kezdjük, ahova az oszlopok nevét, azaz az adatmezők nevét írjuk ki.

```
<TR>
    <% For Each objField In objRecordSet.Fields %>
        <TD ALIGN="center"><B>
<SPAN class=head3><%=objField.Name%></SPAN></B></TD>
    <% Next %>
        <TD COLSPAN="2" ALIGN="center"><B>
<SPAN class=head3>Művelet</SPAN></B></TD>
</TR>
```

Ezután kerül sor a tábla első lapjának a megjelenítésére. A rekordok adatait szövegmezőkbe jelenítjük meg, kivéve az `IsAdmin` mezőt, melynek értékét választógombbal adjuk meg. Az adatok után minden sorban két parancsgombot helyezünk el **"Frissíteni"**, illetve **"Törölni"** felirattal.

```
<TR>
<%
intFirstRecordOnPage = 1 + (intPage-1)*intPageSize
For Each objField In objRecordSet.Fields
%>
<TD>
<%
  If objField.Value="" Then
    Response.Write "<INPUT TYPE=text VALUE="""" & _
    " SIZE=20 NAME=" & _
    objField.Name & "_" & _
    objRecordSet.Fields("UID") & ">"
  Else
    If objField.Name = "UID" Or _
       objField.Nam e= "Email" Or _
```

```
        objField.Name = "RegisteredOn" Then
        Response.Write "<INPUT TYPE=text VALUE=" & _
        objField.Value & " SIZE=20 NAME=" & _
        objField.Name & "_" & _
        objRecordSet.Fields("UID") & _
        " READONLY>"
      Else
        If objField.Name = "IsAdmin" Then
          Dim checked
          If objField.Value Then
            Checked = " CHECKED "
          Else
            checked=""
          End If
          Response.Write _
          "<CENTER><INPUT TYPE=radio VALUE=" & _
          objField.Name & "_" & _
          objRecordSet.Fields("UID") & _
          " SIZE=20 NAME=" & _
          objField.Name & checked & _
          " ></CENTER>"
        Else
          Response.Write "<INPUT TYPE=text VALUE=" & _
          objField.Value & " SIZE=20 NAME=" & _
          objField.Name & "_" & _
          objRecordSet.Fields("UID") & ">"
        End If
      End If
    End If
%>
</TD>
<%
Next
%>
<TD>
<%
Response.Write _
"<INPUT TYPE=submit VALUE=Frissíteni NAME=update_" & _
objRecordSet.Fields("UID") & " DISABLED>"
%>
</TD>
<TD>
<%
Response.Write _
"<INPUT TYPE=submit VALUE=Törölni NAME=delete_" & _
objRecordSet.Fields("UID") & " DISABLED>"
%>
</TD>
</TR>
```

Az adatokat tartalmazó szövegmezők nevét a következőképpen adjuk meg:

```
objField.Name & "_" & objRecordSet.Fields("UID").
```

Más szóval, egy szövegmező neve a megfelelő adatmező nevéből és a szóban forgó rekord azonosítószámából tevődik össze. Ha, teszem azt, a rendszergazda rekordszáma 15, akkor az email-címe az `Email_15` nevű szövegmezőben jelenik meg. A `Request.Form` gyűjteményben ez a *név=érték* pár tehát így fog szerepelni: `Email_15=admin@webhelyem.net` és a többi *név=érték* pártól az `&` karakterrel lesz elválasztva. A **"Frissíteni"**, illetve **"Törölni"** `submit` típusú parancsgombok neve is hasonló:

```
"update_" & objRecordSet.Fields("UID"),
```

illetve

```
"delete_" & objRecordSet.Fields("UID").
```

Ha például a rendszergazda a 25 rekordszámmal rendelkező tag adatait akarja módosítani és a megfelelő **"Frissíteni"** parancsgombra kattint, akkor a `Request.Form` gyűjtemény a következő karakterláncot tartalmazza: `"update_25=Friss%EDteni"` és – ami igen fontos – többször nem jelenik meg az `"update"` karaktersor a `Request.Form` által visszaadott karakterláncba, a `"delete"` karaktersor pedig egyáltalán nincs jelen ebben a karakterláncban. Ha a **"Törölni"** gombot nyomjuk meg, akkor a dolgok hasonlóképpen történnek, csupán az `"update"` és a `"delete"` szerepe cserélődik fel.

A fentiekből kiderül, hogy elég a `Request.Form` gyűjtemény által visszaadott karakterláncban az `"update"` vagy a `"delete"` karakterlánc után kutatni. Kettőjük közül csak az egyik szerepelhet a karakterláncban, vagy egyik sem. Az utóbbi esetben nem kell az adatokat frissíteni vagy törölni. Az előbbi esetben viszont pontosan tudjuk, hogy adatfrissítésről vagy adattörlésről van szó.

```
strRequestStr = Request.Form

If InStr(1,strRequestStr,"delete") <> 0 Then
    ' ha pos1 <> 0, akkor adattörlés
    strAction = "DELETE"
    pos1 = InStr(1,strRequestStr, "delete")
    pos2 = pos1+Len("delete")
    pos3 = Instr(pos1, strRequestStr, "=")
    ' a törlendő rekord száma
    uid = Mid(strRequestStr, pos2 + 1, pos3 - pos2 - 1)
    ' a rendszergazda rekordszáma
    uidAdmin = Mid(Request.Form("IsAdmin"), Len("IsAdmin") + 2)
    If uid = uidAdmin Then
        strMessage = "<SCRIPT LANGUAGE=VBScript>" & _
```

```
            "alert " & _
            """A rendszergazdát nem lehet törölni!""" & _
            "</SCRIPT>"
            strAction="DISPLAY"
        End If
    Elseif InStr(1, strRequestStr, "update") <> 0 Then
        ' ha pos1 <> 0, akkor adatmódosítás
        strAction = "UPDATE"
        pos1 = InStr(1, strRequestStr, "update")
        pos2 = pos1 + Len("update")
        pos3 = Instr(pos1, strRequestStr, "=")
        ' a módosítandó rekord száma
        uid = _
        Mid(strRequestStr, pos2 + 1,pos 3 - pos2 - 1)
        ' a rendszergazda rekordszáma
        uidAdmin = _
        Mid(Request.Form("IsAdmin"),Len("IsAdmin") + 2)
            If uid = uidAdmin Then
                strMessage = _
                "<SCRIPT LANGUAGE=VBScript>" & _
                "alert " & _
                """A rendszergazda adatait nem lehet" & _
                " módosítani!""" & _
                "</SCRIPT>"
                strAction = "DISPLAY"
            End If
            If _
             Len(Request.Form("Password_" & uid)) < 5 Then
             strMessage = "<SCRIPT LANGUAGE=VBScript>" & _
             "alert " & _
             """A jelszó legalább öt karakterből" & _
             " kell álljon!"" " & _
             "</SCRIPT>"
              strAction = "DISPLAY"
            End If
    Else
        pos1 = 0
        ' ha pos1 = 0, akkor megjelenítés
        strAction = "DISPLAY"
    End If
```

Sőt, ennél többet is tudunk. A **"Frissíteni"** vagy **"Törölni"** gomb neve a frissítendő vagy törlendő rekord azonosítószámát is megadja. A törlés esetében ez elég a művelet elvégzésére, a frissítés esetében ki kell olvasnunk a `Request.Form` által visszaadott karakterláncból az új adatokat és minden készen áll a frissítésre. Az adatok törlését, illetve módosítását két eljárásra bízzuk:

```
Sub DeleteRecord(uid)
    Dim strSQL
    strSQL="DELETE * FROM members WHERE UID=" & _
    CInt(uid)
```

```
    GetRS strSQL
End Sub

Sub UpdateRecord(uid, strFirtsName, strSecondName, _
strPassword)
    Dim strSQL
    Dim simpleQuote

    simpleQuote = chr(39)
    strSQL="UPDATE members SET FirstName = " & _
        simpleQuote & _
        strFirtsName & simpleQuote & ", " & _
        "SecondName = " & simpleQuote & _
        strSecondName & simpleQuote & ", " & _
        "Password = " & simpleQuote & _
        strPassword & simpleQuote & " " & _
        "WHERE (members.UID)=" & _
        CInt(uid)
    GetRS strSQL
End Sub
```

Vegyük most sorra a többi <INPUT TYPE=submit> elemet. Ezek NAME attribútuma egységesen "page" . Négyet egy sorban találunk, míg az utolsó a tábla alján van elhelyezve. Ezek VALUE attribútuma rendre "<<-", "<-", "->", "->>" és "Újra letölteni", és mint gyanítható a táblát lapozására használjuk.

```
<TD><INPUT TYPE="hidden" NAME="intPage"
VALUE="<%=intPage%>"></TD>
<TD>
<%
If intPage = 1 Then
    Response.Write "<INPUT TYPE =" & _
    "submit NAME=page VALUE=""&lt;&lt;-"" DISABLED>"
Else
    Response.Write "<INPUT TYPE =" & _
    "submit NAME=page VALUE=""&lt;&lt;-"">"

End If
%>
</TD>
<TD>
<%
If intPage = 1 Then
    Response.Write "<INPUT TYPE=" & _
    "submit NAME=page VALUE=""&lt;-"" DISABLED>"
Else
    Response.Write "<INPUT TYPE=" & _
    "submit NAME=page VALUE=""&lt;-"">"
End If
%>
```

```
</TD>
<TD>
<%
If intPage = intPageCount Then

    Response.Write "<INPUT TYPE=" & _
    "submit NAME=page VALUE=""-&gt;"" DISABLED>"
Else
    Response.Write "<INPUT TYPE=" &_
    "submit NAME=page VALUE=""-&gt;"" >"
End If
%>
</TD>
<TD>
<%
If intPage = intPageCount Then
    Response.Write "<INPUT TYPE=" & _
    "submit NAME=page VALUE=""-&gt;&gt;"" DISABLED>"
Else
    Response.Write "<INPUT TYPE=" & _
    "submit NAME=page VALUE=""-&gt;&gt;"" >"
End If
%>
</TD>
```

A `Request.Form` gyűjtemény csak akkor tartalmaz egy *név=érték* párt, melynek *név* része `page`, ha a "lapozó gombok" egyikét megnyomtuk. Tehát, a `Request.Form` gyűjteményben a `"page"` karakterlánc után kutatva, könnyen rájövünk, kell-e vagy nem lapozni és ha igen, akkor mennyit és merre: a legelső, az előző, a következő vagy a legutolsó lapra. Az ötödik "lapozó gomb" nem lapozást, hanem az aktuális lapon megjelenített rekordok számát módosítja vagy hagyja helyben. A rekordok számát a `<SELECT NAME="pagesize">` listából választhatjuk ki

```
<% Dim i
For i = 1 To 10
    If intPageSize = i Then
        Response.Write "<OPTION SELECTED>" & i & _
                       "</OPTION>"
    Else
        Response.Write "<OPTION>" & i & "</OPTION>"
    End If
Next
%>
```

és a `Request("pagesize")` értékének figyelembevételével adjuk át az `intPageSize` változónak:

```
If Request("pagesize")<>"" Then
    ' rekord/lap
    intPageSize = CInt(Request("pagesize"))
```

```
Else
    ' alapértelmezés 3 rekord/lap
    intPageSize = 3
End If
```

Ezzel befejeztük a `utils/showmembers.asp` lap tárgyalását.

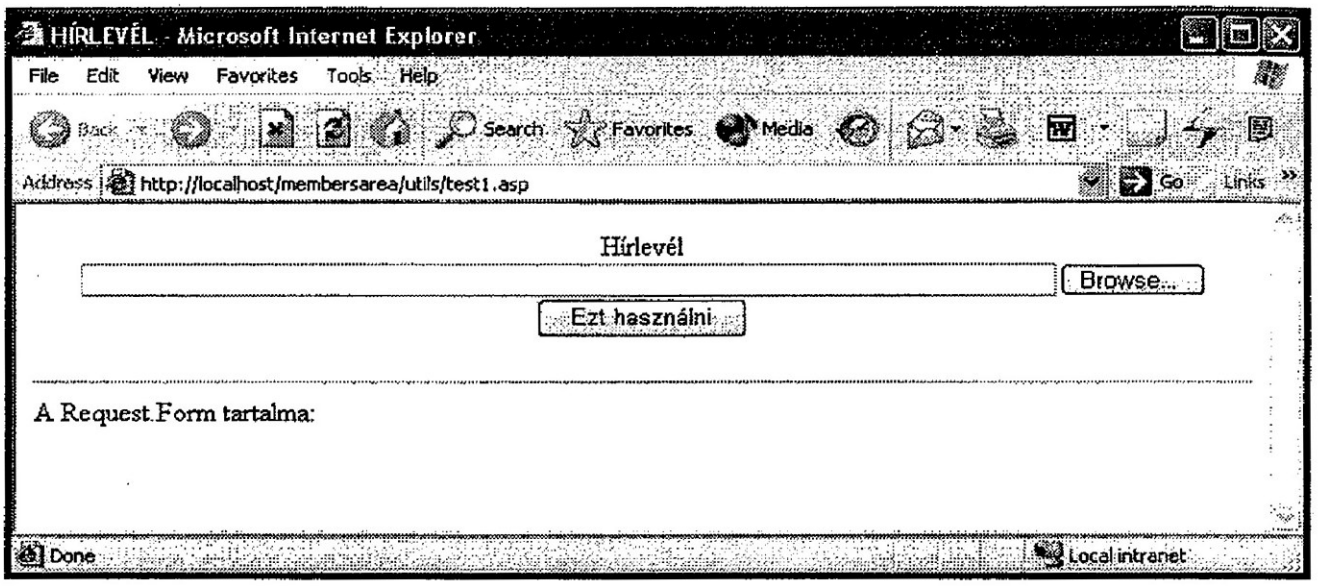
6.9 A hírlevelek elküldése

A hírleveleket a `utils/mailer.asp` állománnyal szerkesztjük meg és küldjük el. Ezt a lapot a `protected/default.asp` lapon elhelyezett "Hírlevelet küldeni" kapcsolat segítségével érjük el. A kapcsolat a `protected/newletters.asp` lapra irányítja a látogatót, ahonnan automatikusan a rendszergazda belépőlapjára (`utils/loginadmin.asp`) kerül. Az előbb láttuk, hogy a beadott adatok helyességét a `utils/checkadmin.asp` ellenőrzi és a rendszergazdát ezúttal, a `protected/newletters.asp` lapon `"newsletters"` értékkel eltátott `Session("AdminAction")` munkamenetszintű változó alapján, beengedi a `utils/mailer.asp` lapra. A `utils/checkadmin.asp` lap a `utils/mailer.asp` lapot egy lekérdezési karakterlánccal (*query string*) hívja meg, mely a `status` változót `1`-re állítja.

A `utils/mailer.asp` lap legelőször ezt a változót kérdezi le és értékének függvényében három különböző eljárást hív meg: `DisplayFormToGetMessageFile`, `DisplayFormToEditMessage` és `SendEmailAndDisplayReport`.

Itt álljunk meg egy pillanatra és nézzük meg, milyen lehetőségei vannak a rendszergazdának. A rendszergazda előre megfogalmazott hírlevelet vagy *online* szerkesztett hírlevelet küldhet el. Az első esetben lehetősége van az előre megfogalmazott hírlevelet, elküldése előtt esetleg még módosítani. Ami minket itt közelebbről érdekel, az az előre megfogalmazott hírlevél. A rendszergazda egy szövegállományba írhatja az elküldendő hírlevelet, elmentheti a saját gépére (ami nem kell, hogy azonos legyen a webszerverrel) és egyszerűen "felküldi" a szövegállományt az alkalmazásnak a szerverre. Hogyan történik ez? Erre a kérdésre keressük a továbbiakban a választ.

A szükséges tudnivalókat egy leegyszerűsített példa kapcsán fogjuk megismeri. Nézzük meg a következő egyszerű ASP `utils/test1.asp` lapot! A 6.5. ábrán láthatjuk a lapot a böngészőben.



6.5. ábra A test1.asp lap a böngészőben

A forráskódja igen egyszerű: egy `<FORM>` elemből áll, mely csupán egy `<INPUT TYPE=file>` és egy `<INPUT TYPE=submit>` vezérlőt tartalmaz:

```
<FORM ACTION=test1.asp METHOD=post ID=form1 NAME=form1>
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
<TR ALIGN=center>
  <TD>Hírlevél</TD>
</TR>
<TR ALIGN=center>
  <TD>
        <INPUT TYPE=file SIZE=85 NAME=fFile >
  </TD>
</TR>
<TR ALIGN=center>
  <TD >
        <INPUT TYPE=submit NAME=submit
        VALUE="Ezt használni" ID=Submit1>
  </TD>
</TR>
</TABLE>
</FORM>
<BR>
<HR>
<%
Response.Write "A Request.Form tartalma:" & "<BR>"
Response.Write Request.Form
%>
```

A `<FORM> ACTION` attribútuma ugyancsak a `test1.asp` lapra utal, a `METHOD` attribútuma pedig `POST`. Ez azt jelenti, hogy a továbbított adatok a `Request.Form` gyűjteménybe kerülnek. Most írjunk egy egyszerű szövegállományt és mentsük el a leme-

zünkre `hirlevel.txt` név alatt. Én a következőkben ezt a háromsoros állományt fogom használni:

```
Ez egy egyszerű próba-hírlevél első sora.
Ez egy egyszerű próba-hírlevél második sora.
Ez egy egyszerű próba-hírlevél utolsó sora.
```

Hívjuk le a `test1.asp` lapot, keressük meg a gépünkön a `hirlevel.txt` állományt és nyomjuk meg az **"Ezt használni"** feliratú gombot. A lapon megjelenik a `Request.Form` gyűjtemény tartalma:

```
fFile=E%3A%5Cusers%5Chatvany%5C_Code%5CASP.Classic%5CSamples%5Cmembers
area%5Cutils%5Chirlevel.txt&submit=Ezt+haszn%E1lni
```

Könnyen felismerhetjük a két HTML vezérlő nevét: `fFile` és `submit` és a megfelelő értékeket, úgy ahogy ez *név=érték* párok formájában a `Request.Form` gyűjtemény tagjainál szokásos. Itt két ilyen párral van dolgunk, köztük a `&` elválasztó karakter. Ha figyelembe vesszük az URL kódolást (`%3A` = `:`, `%5C` = `\`, `+` = *szóköz* és `%E1` = `á`), akkor világos, hogy az első pár értéke a felküldött állomány elérési útja, a másodiké pedig a használt parancsgomb felirata, azaz `value` attribútumának értéke. Sajnos a böngészőnk nem küldi fel a szerverre az állomány tartalmát. Ahhoz, hogy ez is felkerüljön a `<FORM>` elem `ENCTYPE` attribútuma, melynek alapértelmezésben `"application/x-www-form-urlencoded"` az értéke, `"multipart/form-data"` kell lennie. Módosítsuk a `utils/test1.asp` lapot ezzel a kiegészítéssel. Az eredményt a `utils/test2.asp` állomány (ügyeljünk arra, hogy a `<FORM>` elem `ACTION` attribútumát is megváltoztassuk!). Töltsük le a `utils/test2.asp` állományt, válasszuk ki ismét a `hirlevel.txt` állományt és nyomjuk meg az **"Ezt használni"** feliratú gombot. Az eredmény lehangoló: ha a `<FORM>` elem `ENCTYPE` attribútuma `"multipart/form-data"`, akkor a böngésző nem tölti ki a `Request.Form` gyűjteményt, hanem az adatokat a HTML fejlécben küldi fel a szerverre. Hogy ezeket az adatokat elérhessük, a `Request` objektum `TotalBytes` tulajdonságát és a `BinaryRead` metódusát kell használnunk, amivel a 2. fejezetben találkoztunk. A `TotalBytes` tulajdonság csak olvasható és a kliens által a `POST` metódussal elküldött bájtok számát adja meg. A `BinaryRead` metódus szintaxisa a következő:

```
változó = Request.BinaryRead(bájtszám)
```

A metódus *bájtszám* bájtot olvas a *változó*ba. A *változó*ba nem egy karakterlánc, hanem egy úgynevezett biztonságos bájttömb (*safe array of bytes*) kerül. A tulajdonság és a metódus biztonságos alkalmazására elegendő tudni, hogy a `BinaryRead` metódus nem ad vissza egy karakterláncot és ennélfogva visszatérő értékére nem alkalmazható semmilyen függvény, ami a karakterláncokra vonatkozik. Hogy ezt belás-

suk, bővítsük kísérletünket az új tulajdonsággal és metódussal. Az így kapott állományt nevezzük el `utils/test3.asp`-nek. Az állomány forráskódjában készakarva, de *helytelenül*(!) használom a `Response.Write` metódust: ugyanis ezt a metódust csak karakterláncokra alkalmazhatjuk:

```
<%
intByteCount
Dim binArray
' beolvasott bájtok száma
intByteCount = Request.TotalBytes
' beolvasott bájtsor
If intByteCount<>0 Then
    binArray = Request.BinaryRead(intByteCount)
End If
Response.Write "Beolvasott bájtok száma: " & _
intByteCount & "<BR>"
Response.Write "Beolvasott bájtsor: " & "<BR>"
' csak a kísérlet kedvéért, különben helytelen
Response.Write binArray
%>
```

Kíváncsiságból nézzük meg az eredményt. Az én `hirlevel.txt` állományommal ez így néz ki (a kérdőjelek sorát lerövidítettem):

```
Beolvasott bájtok száma: 494
Beolvasott bájtsor:
????????????????????????????????
```

Ha valaki kíváncsi lenne a kérdőjeleket megszámolni, akkor 494/2 (egész számú osztás) darabot találna. Amint ezt a 2. fejezetből tudjuk, Visual Basic Script a karakterek UNICODE szerinti kódolását használja, ahol minden karakternek két bájt felel meg. A `Response.Write` metódus UNICODE szerint kódolt karaktersort vár, de (helytelenül) egy 494 bájtól álló "nyers" bájtsort kap. A `Response.Write` metódus a kapott bájtokat kettesével próbálja értelmezni, de rendszerint nem akad semmilyen "értelmes" UNICODE kódra és ilyenkor egy kérdőjelt ad vissza. Nagyon ritkán előfordul, hogy a beolvasott két bájt megegyezik egy karakter UNICODE kódjával. Ilyenkor megjelenik ez a karakter, de ez nagyon ritkán a helyes karakter. A következtetés, ami innen adódik világos: így nem jutunk előre. Át kell alakítanunk a "nyers" bájtsort karakterlánccá, hogy "láthassuk" és hogy dolgozhassunk vele.

Két egyszerű függvényt szoktunk erre a célra használni. Definíciójuk és használatuk a `utils/test4.asp` állományban látható.

```
<%
' segédfüggvények
' átalakít egy bájtsort karakterlánccá
Function byte2str ( bin )
    Dim i, bytebuf
```

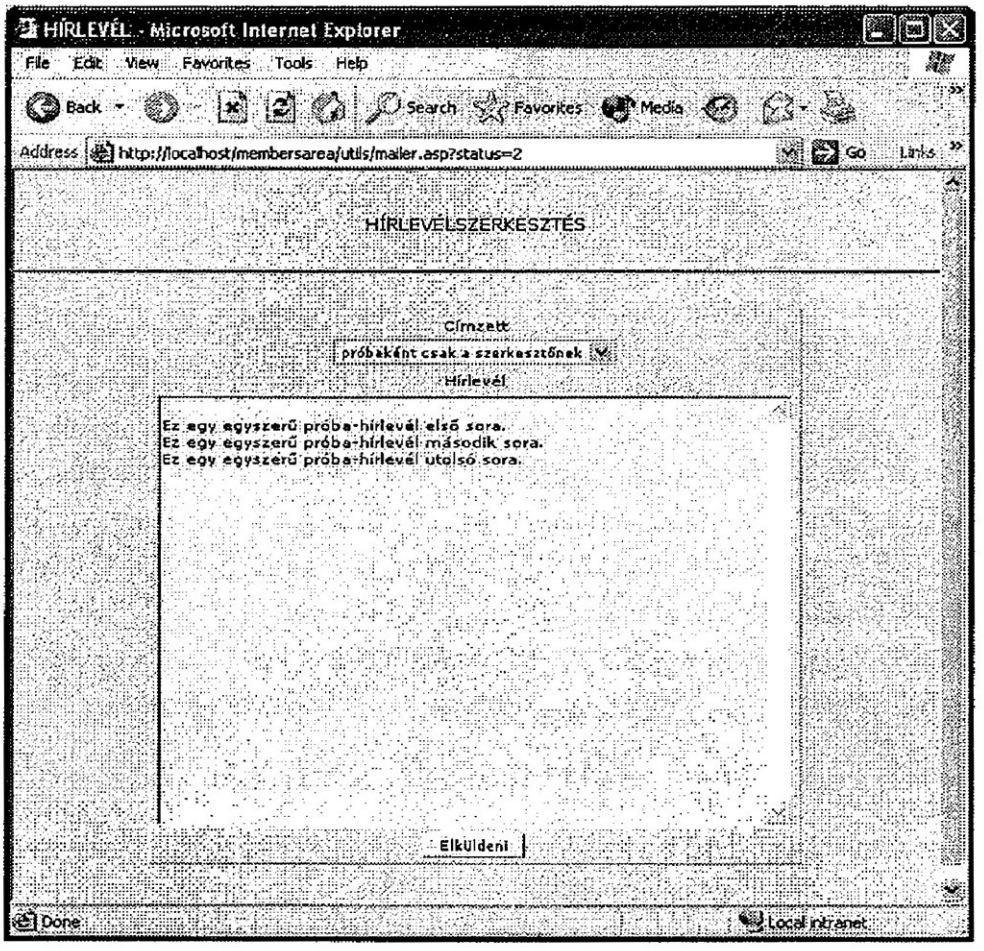
```
    For i = 1 To LenB(bin)
        bytebuf = bytebuf & Chr(AscB(MidB(bin, i, 1)))
    Next
    byte2str = bytebuf
End Function

' átalakít egy karaktersort bájtsorrá
' - itt nem használjuk: csak a teljesség kedvéért
Function str2byte ( str )
    Dim i, strbuf
    For i = 1 To Len(str)
        strbuf = strbuf & ChrB(AscB(Mid(str, i, 1)))
    Next
    str2byte = strbuf
End Function
%>
```

A felhasznált `LenB`, `MidB` és `AscB` függvények a hasonnevű karakterláncoknál használt függvényeknek felelnek meg, azzal a különbséggel, hogy karakterek helyett bájtokkal dolgoznak. Az így karakterlánccá átalakított bájtsort most megjeleníthetjük a `Response.Write` metódus segítségével. A már ismert `hirlevel.txt` állományt és a `utils/test4.asp` lapot használva, az eredmény ezúttal ez (az egész egyetlen egy hosszú sorban):

```
-----------------------------7d213e204052e Content-Disposition: form-
data; name="fFile";
filename="E:\users\hatvany\_Code\ASP.Classic\Samples\membersarea\utils
\hirlevel.txt" Content-Type: text/plain Ez egy egyszerű próba-hírlevél
első sora. Ez egy egyszerű próba-hírlevél második sora. Ez egy
egyszerű próba-hírlevél utolsó sora. -----------------------------
7d213e204052e Content-Disposition: form-data; name="submit" Ezt
használni -----------------------------7d213e204052e--
```

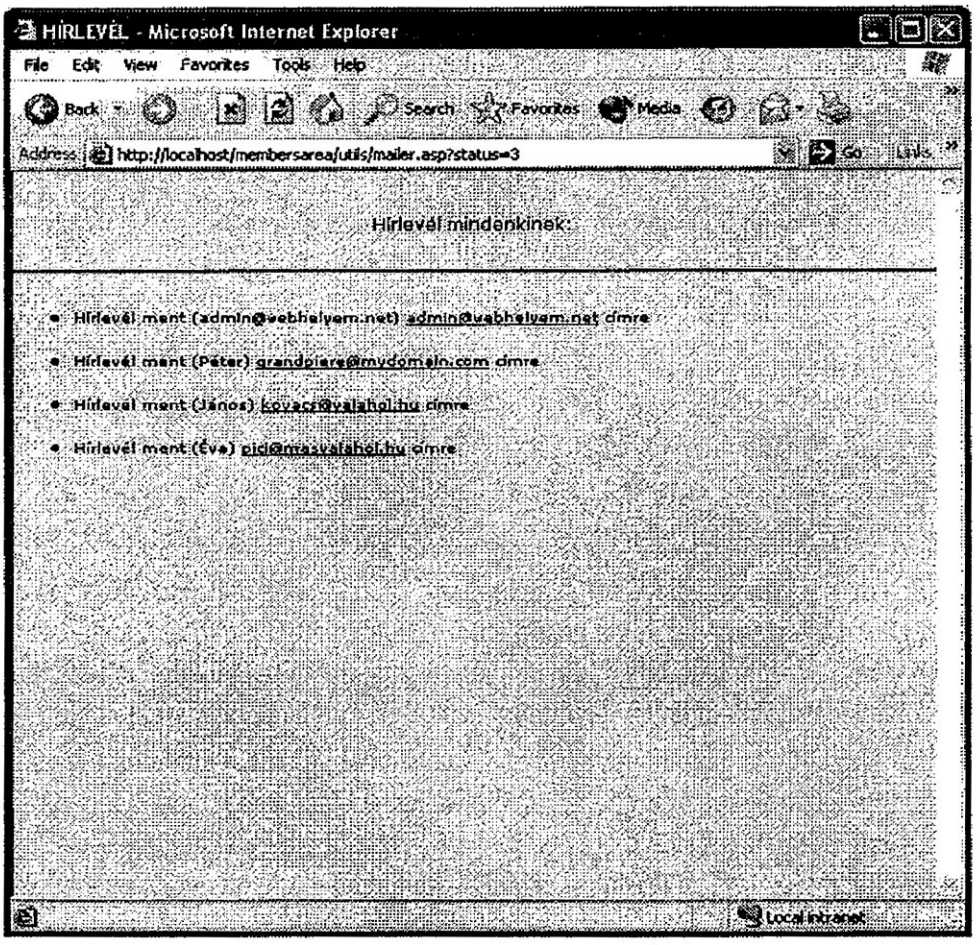
Több dolgot vehetünk észre. A legfontosabb számunkra, hogy a karakterlánc tartalmazza a `hirlevel.txt` állomány szövegét. Más szóval feltöltöttük az állományt a szerverre! Aki ezzel a témával kapcsolatban többet szeretne tudni, az meglátogathatja az **The Internet Engineering Task Force** erre vonatkozó honlapját (http://www.ietf.org/rfc/rfc1867.txt?number=1867). Ezen túl, vegyük észre, hogy a karakterláncban többször megismétlődik egy karaktersor. A mi esetünkben ez a karaktersor -----------------------------7d213e204052e. Egy elválasztó (*separator*) elemről van szó, melyet a böngészőnk állít elő, mindig más, de egy felküldött adatcsomagon belül mindig ugyanaz. Ezt az elválasztó elemet felhasználva, ki tudjuk szűrni a karaktersorból a szövegállomány tartalmát. A felküldött állomány MIME típusát a `Content-Type:` adja meg. Ezt felhasználva, korlátozhatjuk a szerverre felküldhető állományok típusát. A mi példaalkalmazásunkban, amihez e kis kitérő után máris visszatérünk, csak szövegállományokat fogadunk el.

A `DisplayFormToGetMessageFile` és `DisplayFormToEditMessage` eljárás az előbbiek fényében most érthetőnek kell lennie. Miután kiválasztottunk egy szövegállományt, lehívjuk a `utils/mailer.asp` lap "második alakját" (6.6. ábra).



6.6. ábra A hírlevél elküldése

Itt egy `<TEXTAREA>` vezérlőbe láthatjuk a hírlevél szövegét, amit esetleg még módosíthatunk. Ha a `utils/mailer.asp` lapon megnyomjuk az **"Ezt használni"** feliratú gombot anélkül, hogy egy állományt választottunk volna ki, akkor a `mess` változó üres marad. Ebben az esetben a `utils/mailer.asp` lap "második alakján" megjelenő `<TEXTAREA>` vezérlőbe beadhatunk egy szöveget. Mielőtt a rendszergazda elküldené a hírlevelet, a lapon elhelyezett választólistával eldöntheti, hogy csak egy próbalevélről van-e szó vagy elküldi a hírlevelet mindegyik regisztrált tagnak. Az első esetben csak a rendszergazda kapja meg a próbalevelet. A hírlevél elküldése után a `utils/mailer.asp` lap "harmadik alakján" megjelenik egy szűkszavú beszámoló az elküldött levelekről. A szkript hátramarad része már ismert.



6.7. ábra A beszámoló a hírlevél elküldéséről

6.10 Befejező megjegyzések

Hangsúlyoznom kell, hogy alkalmazásunk távolról sem olyan biztonságos, mint a *Secure Sockets Layer* (SSL) protokollt használó alkalmazások, de célunkra – gondolom – megfelel. Erről a protokollról az érdekelt Olvasó a következő cím alatt többet talál: http://developer.netscape.com/docs/manuals/security/sslin/contents.htm.

Az alkalmazást többféleképpen lehet bővíteni. A regisztráló lapon például, megadhatjuk a lehetőséget, hogy az új látogató kiválaszthassa, HTML vagy csak sima szöveges hírlevelet szeretne kapni. Ez több módosítást vonna maga után: a regisztrálásnál a látogató megadhatná a kívánt hírlevélformát, az adatok módosításánál meg kell adni a lehetőséget a regisztrált tagnak, hogy a hírlevélformát is módosíthassa és a hírleveleket elküldő rutinokat ennek megfelelően kellene átalakítani. Egy másik elképzelhető bővítés egy napló (*logfile*) lenne, amit csak a rendszergazda kezelhetne.

Hogyan tovább?

Minden tanfolyam elvégzése után, illetve minden tankönyv elolvasása után a fenti kérdés jogos és – amint ez egyszerű kérdéseknél mindig lenni szokott – a válasz összetett. Ennek ellenére nem akarok adós maradni a válasszal.

Ha a kedves Olvasó munkaköre olyan, hogy még egy pár évig ASP 3.0 alkalmazásokat kell készítenie, akkor érdemes az ASP speciális fejezeteivel megismerkednie. Ide tartozik például az ActiveX Data Objects (ADO) adatkezeléssel foglakozó programozási interfész vagy az Extensible Markup Language (XML) megjelenítési és dokumentumtárolási formátum.

Érdemes a világhálón is idevágó információt keresni. Sokszor találunk olyan megoldásokat, melyek ha nem is oldják meg az éppen a megoldásra váró problémánkat, közelebb visznek a keresett megoldáshoz. A webhelyemen (www.hatvany-online.net) a kedves Olvasó három oldalon talál ASP-vel kapcsolatos információt. A www.hatvany-online.net/asplinks/asplinks1.htm oldal egy hivatkozás-gyűjteményt tartalmaz a legismertebb ASP webhelyekkel. A www.hatvany-online.net/asplinks/asplinks2.htm és a www.hatvany-online.net/asplinks/asplinks3.htm oldalak a DevelopersDex.com (www.developersdex.com), illetve a DevASP.com (www.devasp.com/) webhelyek automatikusan felfrissített tartalomjegyzékét jelenítik meg.

Három évvel ezelőtt Microsoft bevezette a .NET (dotnet) név alatt ismert új technológiát. Az új technológia új alapokra helyezi a webalkalmazások fejlesztését is. A .NET technológiában az immár klasszikus ASP-t az ASP.NET váltja fel. Ha olyan a munkakörünk, hogy rövid időn belül bevezetésre kerül a .NET technológia, akkor érdemes foglalkozni a gondolattal, hogy – felhasználva a most megszerzett „klasszikus" ASP ismereteket – nem előnyösebb-e már most megismerkednünk az új technológiával.

Sajnos pillanatnyilag kevés anyag jelent meg magyarul a .NET technológiáról. Egy bevezető cikket, ami az ASP.NET rövid ismertetésére is kitér, az első magyar fejlesztői portálon, a www.prog.hu-n *„Bevezetés a .NET-be: Hét variáció egy Kernigham-Ritchie-témára"* címmel közöltem (http://www.prog.hu/article.php?aid=860). Ezt egy jó kiindulópontnak tartom a .NET technológia megismerésére.

Szándékomban áll, hogy a 2004. tavasz végére befejezzem a *„Bevezetés az ASP.NET programozásba"* munkacímmel jelzett könyvemet, ami ugyanennél a Kiadónál fog megjelenni. Remélem Olvasóim közül sokkal fogok az új könyvem lapjain találkozni. Ugyanis ennek a könyvnek is a szemléltető anyagát folyamatosan felteszem a

webhelyemre (http://www.hatvany-online.net/aspdotnet) és szívesen fogadok építő jellegű észrevételeket, megjegyzéseket, tanácsokat.

A legközelebbi virtuális találkozásig sikeres kódolást kívánok minden kedves Olvasómnak!

Irodalomjegyzék

[1] *** **Microsoft Developer Network (MSDN) Library,**
Active Server Pages (ASP),
http://msdn.microsoft.com/nhp/default.asp?contentid=28000522

[2] *Richard Anderson et al.*:
ASP 3.0 Programmer's Reference,
Wrox Press, 2000, ISBN 1861003234

[3] *Richard Anderson, Alex Homer, Simon Robinson*:
Beginning Components for ASP,
Wrox Press, 1999, ISBN 1861002882

[4] *David Buser et al*:
Beginning Active Server Pages 3.0,
Wrox Press, 1999, ISBN 1861003382

[5] *Alex Homer*:
Professional ASP Techniques for Web Masters,
Wrox Press, 2000, ISBN 1861001797

[6] *John Kauffman*:
Beginning ASP Databases,
Wrox Press, 1999, ISBN 1861002726

[7] *Simon Robinson et al.*:
Professional ASP Data Access,
Wrox Press, 2000, ISBN 1861003927

Tárgymutató

%

%>, 42

.

.NET, 305

<

<%, 42, 93
<%@ ... >, 94
<BODY>, 96
<FORM>, 82, 110, 290, 298
<HR>, 174
<HTML>, 96
<implements>, 232
<INPUT type="file">, 82
<INPUT TYPE=submit>, 295
<INPUT>, 290
<METADATA>, 217
<method>, 232
<OBJECT>, 44, 60, 120, 167, 178, 183, 196
<property>, 232
<registration>, 230, 231
<SCRIPT ...> utasítás, 34
<script>, 180, 231, 232
<scriptlet>, 230
<SELECT NAME="pagesize">, 296
<SELECT>, 206
<TEXTAREA>, 302

A

Abandom metódus, 73
About, 184
AbsolutePage, 291
ACTION, 110, 207
Active protocol, 100
Active Server Pages, 5
ActiveX Data Object, 265
adatforrás, 276
AddHeader, 99
adHomePageURL, 168, 169
Administrative Tools, 9
ADO, 265, 305
AdRotator, 165, 167, 170
adURL, 168
alkalmazásszintű objektum, 44
ALL_HTTP, 81
ALL_RAW, 81
ANSI kód, 79
AppendToLong metódus, 100
Application name, 22
Application objektum, 41, 54, 56, 61
Application_OnEnd, 44, 59
Application_Onstart, 54
Application_OnStart, 57
application-level object, 44
ASP, 5
ASP 3.0, 1
ASP belső objektumai, 41
ASP kód, 42
ASP változó, 43
ASP végrehajtási direktíva, 29
ASPClassic menü, 27
ASPClassis webalkalmazás, 47
ASPCode, 114
ASPError, 114
ASPError objektum, 52, 113, 116, 118
AtEndOfLine, 157
AtEndOfStream, 157
átirányító, 169
átjáró, 102
attrib, 142
Attribute, 141, 151
AvailableSpace, 134
availHeight, 179
availWidth, 179

B

behavior, 179
belső objektum, 6
BinaryRead metódus, 77, 299
biztonsági bizonylat, 90
biztonságos bájttömb, 299
biztonságos csatorna, 76
biztonságos tömb, 77
Body, 210
BodyFormat, 210
Border, 169, 170
BrowserType, 165, 175, 177, 181
BrowserType komponens, 179
Buffer, 94, 106
bufferDepth, 179
BuildPath metódus, 127

C

CA, 90
CashControl, 102, 103
cashing, 33, 102
Category, 115
CDFS, 136
CDO, 208
CDO for NT systems, 208
CDONTS, 166, 208
CDOSYS, 166, 213, 280
Certificate Authory, 90
CGI, 3, 5
cgi-bin mappa, 3
Charset, 103
ChooseContent, 174
class identifier, 217
ClassID, 231
Clear, 94, 95
Clickable, 169, 170
clientCaps, 179
Close metódus, 157
CloseLogFiles metódus, 194
CLSID, 217, 218, 231
Codepage, 68
CODEPAGE direktíva, 30
Collaboration Data Object, 208
colorDepth, 179
Column, 115, 158
COM, 215
Common Gateway Interface, 3
Compact Disk File System, 136
CompareMode, 161
Component Object Model, 215
connectionType, 179
content list, 182
content scheduler, 173
ContentRotator, 165, 173
Contents, 54, 62
context menu, 11
cookieEnabled, 179
cookies, 64, 76
Cookies, 91, 105
Copy metódus, 139, 147
CopyFile metódus, 123
CopyFolder metódus, 123
Count, 61, 161
Counters, 165, 187
cpuClass, 179
CreateFolder metódus, 125
CreateObject metódus, 44, 58, 60, 216, 241, 259
CreateTextFile metódus, 125, 140

D

data driven, 1
data source, 276
DateCreated, 143, 153
DateLastAccessed, 143, 153
DateLastModified, 143, 153
Default Web Site, 17, 19
Degree, 247
Delete metódus, 140, 148
DeleteFile metódus, 124
DeleteFolder metódus, 124
Description, 115, 231
destination, 123, 139, 147
Dictionary objektum, 119, 159
digital certificate, 76
digital security certificate, 90
digitális bizonylat, 76
Domain, 106
domain-név, 105, 205
dotnet, 305
Drive, 144, 154
Drive objektum, 119, 133, 138
DriveExists metódus, 122
DriveLetter, 135

E

egyedi információt kezelő, 165
elfelejtett jelszó, 265
előregyártott objektum, 6
elválasztó, 301
End, 95
eszköztár, 165
Execute metódus, 45
Exists metódus, 160
Expires, 103, 107
ExpiresAbsolut, 103
Extended Properties, 100
Extensible Markup Language, 230, 305

F

FAT, 136
fejlécelem, 81
Fields, 214
File Allocation Table, 136
File objektum, 119, 146, 150
FileExists metódus, 122, 205
FileSystem, 136
FileSystemObject objektum, 49, 119, 122, 127, 132, 215
Flush, 94, 95
Folder objektum, 119, 138, 140, 146
FolderExists metódus, 122
FormDegree, 247
frame, 170
FreeSpace, 134
frissítő lekérdezés, 288
FSO, 120

G

gateway, 102
Get metódus, 188
GET metódus, 83
GetAbsolutePathName metódus, 132
GetAdvertisment metódus, 169
GetAllContent metódus, 174
GetBaseName metódus, 130
GetDrive metódus, 123
GetDriveName metódus, 128
GetExtensionName metódus, 130
GetFile metódus, 123
GetFolder metódus, 123
GetLastError metódus, 52, 113
GetListCount metódus, 184
GetListIndex, 184
GetNextDescription, 184
GetNextURL, 184
GetParentFolderName metódus, 132
GetPreviousDescription, 184
GetPreviousURL, 184
GetSpecialFolder metódus, 127, 138
GetTempName metódus, 132
global.asa, 42, 44, 65
globally unique identifier, 217
GUID, 217
gyorskiszolgálás, 102
gyorstár, 33

H

HasAccess metódus, 202
HasKey, 108
header, 81
height, 179
helyi azonosító, 69
Hidden, 142, 152
hirdetésforgató, 165
hírlevelek elküldése, 266
Hits metódus, 200
HTML adatáramlat, 92
HTML data stream, 92
HTML kód, 32, 50, 92
HTML lap, 46
HTMLEncode metódus, 49
HTTP fejléc, 92, 98
HTTP header, 92
HTTP protokoll, 77
HTTP_ACCEPT_LANGUAGE, 81
HTTP_REFERER, 88
HTTPS protokoll, 108

I

IIS, 7, 92
IIS preprocesszora, 114
IISHelp, 16
IISHelp ikon, 11
IISLog, 165, 190, 192
Importance, 210
Increment metódus, 58, 188

InputBox, 221
installable components, 165
instrinsic, 41
intApplicationCounter, 59
InterDev, 24
Internet Explorer, 7, 20
Internet Information Services, 7
  beállítása, 7
  telepítése, 7
Internet Server Application Programming Interface, 4
Internet Services Manager, 9
ISAPI, 4
IsClientConnected, 105
ISM, 9
IsReady, 136
Item, 108, 161

J

javaEnabled, 179
jelszó, 273
jogosultság-ellenőrző, 165
JScript, 1, 5, 17, 32, 119, 150, 218

K

karakterlánc, 276
keret, 170
key, 161
Key, 62
kliens, 14, 215
kliensgép, 105
kliensoldali szkript, 37

L

LANGUAGE direktíva, 30
lapszámláló, 165
lapszintű objektum, 44
LCID, 68
LCID direktíva, 30
lekérdezési karakterlánc, 297
lekérdezési lánc, 272
Line, 115, 158
locale identifier, 69
Lock, 54
Lock metódus, 63
logfile, 169, 303

M

MapPath metódus, 49
markup language, 230
members area, 265
metadata, 120
METHOD, 110
Microsoft Active Server Pages, 1
Microsoft ActiveX Scripting, 218
Microsoft Management Console, 9, 10
Microsoft Web Components, 165
MidB, 78
MIME, 104
MMC, 9, 10, 21, 46
Move metódus, 139, 147
MoveFile metódus, 124
MoveFolder metódus, 124
MSDN, 179
MsgBox, 221
MSWC, 165, 202
Multi-purpose Internet Mail Extension, 104
Multi-purpose Internet Multimedia Extension, 104
munkamenet, 15
munkamenet-hatókörű objektum, 44
munkamenet-változó, 16
munkamenetszintű változó, 16
MyInfo, 165, 196

N

Name, 143, 153
napló, 303
napló formátum, 100
naplókezelő, 165
Netscape Navigator, 52
Netscape Server Application Programming Interface, 4
NextLink, 165, 182, 183, 184
NotePad, 24, 50, 112, 199
Nothing, 144
NSAPI, 4
null, 75

O

OnEnd, 54
OnEndPage metódus, 259
online szerkesztett hírlevél, 297
OnStart, 54
OnStartPage metódus, 262
Open in Browser, 11
OpenAsTextFile metódus, 148
OpenLogFile, 193
OpenTextFile metódus, 126
Option Explicit, 94

Ö

önálló alkalmazás, 15

P

PageCount, 290
PageCounter, 165, 200, 201
PageHit metódus, 201
page-level object, 44
page-level variable, 16
PageSize, 290
pageURL, 182
ParentFolder, 144, 154
Path, 107, 136, 143, 153
PermissionChecker, 165, 202, 204
PICS, 101
platform, 179
Platform for Internet Control Selection, 101
platformfüggetlen, 4
POST metódus, 78, 83, 89
pre-processor, 114
process, 216
ProcessForm metódus, 205
processing directive, 29
PROGID, 218, 237
ProgID mappa, 237
programmatic identifier, 218
Properties menüparancs, 20
protected, 287
provider, 276
publikált weblapok, 9
Publish varázsló, 9

Q

query string, 272, 276, 297
QueryString, 87, 90, 169

R

Random, 273
Random metódus, 205, 208
Read metódus, 156
ReadAll metódus, 157
ReadLine metódus, 156
read-only, 134
ReadOnly, 142
RecordSet objektum, 290, 291
REDIRECT, 169
Redirect metódus, 99, 272
redirector, 169
Registry, 236
regisztrált tagok bejelentkezése, 265
relatív virtuális elérési út, 45
Remove metódus, 160, 162, 188, 201
RemoveAll metódus, 162
Request objektum, 76, 80, 268
Response objektum, 31, 37, 94, 106
Rnd, 273
RootFolder, 136
RUNAT attribútum, 61

S

safe array, 77
safe array of bytes, 299
scheduler, 167
script host, 32
script interpreter, 218
Scripting objektum, 1, 119, 166
Scripting RunTime Library, 119
ScriptingContext objektum, 262
scriptlet, 230
Scriptlet Wizard, 226, 229, 242
scrrun.dll, 120
secure channel, 76
Secure Sockets Layer, 76, 90, 303
Send metódus, 210
SendCDONTSEmail, 209
separator, 301
SerialNumber, 137

Server objektum, 41, 43, 44, 53, 120, 216, 259, 276
ServerTimeout tulajdonság, 53
session, 15, 267
Session objektum, 41, 64
Session_OnEnd, 65
Session_OnStart, 44, 65
SessionID, 68, 75
session-level, 267
session-level variable, 16
session-scope, 44
Set metódus, 188
SetUp metódus, 263
ShareName, 137
ShortName, 143, 153
ShortPath, 143, 153
Size, 143, 153
Skip metódus, 156
SkipLine metódus, 156
skriptértelmező, 218
Source, 116
SSL, 76, 90, 303
stand-alone alkalmazás, 15
StaticObjects, 54, 61
stdin/stdout interfész, 3
SubFolder, 144
Subject, 210
sütik, 64, 76
süti neve, 181
systemLanguage, 179
szabványosított jelölő nyelv, 230
számlálókat kezelő, 165
szerver, 215
szkriptértelmezés, 32
szkriptnyelv, 34
szkriptnyelv értelmező, 218
szolgáltató, 276

T

tagok adatainak módosítása, 266
tagok törlése, 265
TargetFrame, 169, 170
tartalomforgató, 165
tartalomjegyzék, 182
tartalomkapcsoló, 165
tartalomütemező, 173
telepíthető komponensek, 165
TextPad, 200
TextStream objektum, 119, 155
TimeOut tulajdonság, 66
típuskönyvtár, 120
ToDergee, 247
Tools, 165, 205, 206
Total visits, 66
TotalBytes, 76, 77, 80
TotalSize, 134
TRANSACTION direktíva, 30
Type, 143, 153
type library, 120

U

új tagok regisztrálása, 265
UNC, 137
UNICODE, 300
Unified Resource Locator, 17
Universal Naming Convention, 137
universal unique identifier, 217
UnLock, 54
URI Query, 100
URL, 17, 45
URLEncode metódus, 52
userLanguage, 179
UUID, 217

Ü

ütemező, 167

V

Value, 68
Value metódus, 178
Variant, 133, 160
VBS webalkalmazás, 23
VBScript, 1, 5, 17, 24, 32, 119, 133, 150, 218, 222
virtual application, 15
Virtual Directory, 17
Virtual Directory Creation varázsló, 18
Virtual Directory lap, 12
viselkedés, 178
VolumeName, 137

W

W3C, 230
W3C Extended Log File Format, 100
web site, 1
webalkalmazás, 14, 165, 265
webhely, 1, 276
webhirdetések, 165
webkliens, 29
weblap, 15
webszerver, 14, 16
webszolgáltatások, 29
webszolgáltató, 276
webterület, 265
webügyfél, 29
width, 179
Window Script Components, 218
Windows Script Host, 218
World Wide Web Consortium, 230
Write metódus, 50, 76, 79, 92
WriteBlankLines metódus, 157
WriteLine metódus, 157
WriteLogRecord metódus, 193
WSC, 218
WSH, 218, 219

X

XML, 230, 305